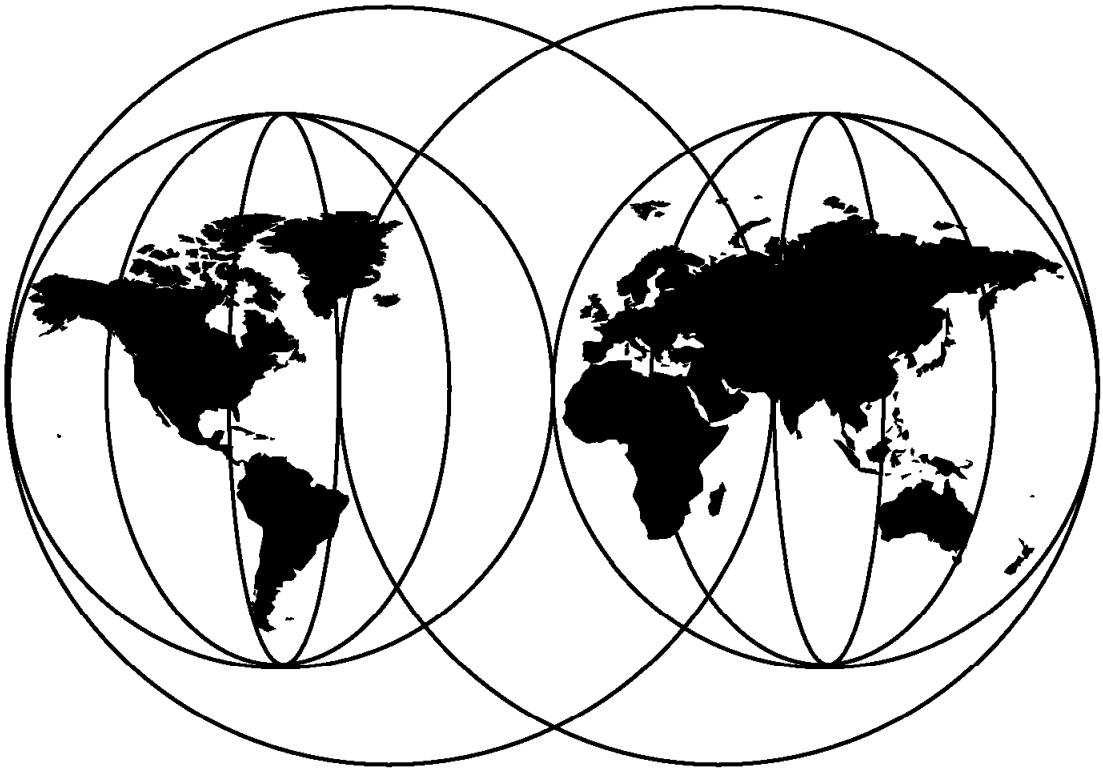
Exploiting Recent CMS Function: A User's Guide to CMS Application Multitasking

*Perry Ruiter ** Holger Woller*



International Technical Support Organization

http://www.redbooks.ibm.com

This book was printed at 240 dpi (dots per inch). The final production redbook with the RED cover will be printed at 1200 dpi and will provide superior graphics resolution. Please see "How to Get ITSO Redbooks" at the back of this book for ordering instructions.

SG24-5164-00

International Technical Support Organization

SG24-5164-00

Exploiting Recent CMS Function: A User′s Guide to CMS Application Multitasking

November 1998

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix D, “Special Notices” on page 105.

First Edition (November 1998)

This edition applies to Virtual Machine/Enterprise Systems Architecture (VM/ESA), Version 2 Release 3.0, Program Number 5654-030, and subsequent releases. As noted in the text, however, the vast majority of the material is equally applicable to all releases of VM/ESA supported at the time the book is written.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
522 South Road
Poughkeepsie, New York 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

Preface

CMS Application Multitasking has been an integral part of CMS since CMS level 9 (VM/ESA Version 1 Release 2.0); however, many CMS programmers, particularly REXX programmers, have not yet learned how to take advantage of the highly significant and useful functions, such as event, queue and timer services, that this suite of APIs provides. The reasons that these functions are often overlooked, even by experienced CMS developers, are varied but include:

- user group presentations about the capabilities have generally focused on writing servers
- the belief that these functions are useful only for multitasking applications
- perceived difficulty using CSLs, especially from REXX

The intent of this redbook is to provide an approachable, introductory user′s guide that addresses these concerns by offering both tutorial information on the capabilities and useful sample programs that installations can readily adapt to meet local needs. To insure that the information presented is accessable to the widest possible audience, the examples provided are written in REXX. We believe, however, that the information presented will also be useful to developers writing in other languages. When you have finished reading this material, you will be able to recognize where these APIs can be usefully applied, both for enhancing existing applications or developing new ones, whether multithreaded or not.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Poughkeepsie Center.

Perry Ruiter works as a Senior Technical Analyst in the VM group at the Government of the Province of British Columbia, Canada. He has worked exclusively with VM for the last 13 years helping users exploit its capabilities.

Holger Woller is responsible for usage concepts in the WorldWide VM Team of IBM. He has 10 years of experience in the VM area, including VM automation and networking of all kinds. He can be reached at: Holger@vnet.ibm.com

The residency that produced this redbook was coordinated by:

Stephen Record
International Technical Support Organization, Böblingen Center

Thanks to the following people for their invaluable contributions to this project:

Jack Crast
IBM Endicott

Arty Ecock
City University of New York

Joe Melligan
IBM Endicott

Steve Shultz
IBM Endicott

Brian Wade
IBM Endicott

Gudrun Wiedemann
International Technical Support Organization, Böblingen Center

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in “ITSO Redbook Evaluation” on page 127 to the fax number shown on the form.
- Use the electronic evaluation form found on the Redbooks Web sites:

  For Internet users `http://www.redbooks.ibm.com/`
  For IBM Intranet users `http://w3.itso.ibm.com/`

- Send us a note at the following address:

  `redbook@us.ibm.com`

Chapter 1. Introduction

CMS application multitasking has been an integral part of CMS since CMS level 9 (VM/ESA Version 1 Release 2.0); however, many CMS programmers, particularly REXX programmers, have not yet learned how to take advantage of the highly significant and useful functions, such as event, queue and timer services, that this suite of APIs provides. The reasons that these functions are often overlooked, even by experienced CMS developers, are varied but include:

- user group presentations about the capabilities have generally focused on writing servers
- the belief that these functions are useful only for multitasking applications
- perceived difficulty using CSLs, especially from REXX

The intent of this redbook is to provide an approachable, introductory user′s guide that addresses these concerns by offering both tutorial information on the capabilities and useful sample programs that installations can readily adapt to meet local needs. To insure that the information presented is accessable to the widest possible audience, the examples provided are written in REXX. We believe, however, that the information presented will also be useful to developers writing in other languages. When you have finished reading this material, you will be able to recognize where these APIs can be usefully applied, both for enhancing existing applications or developing new ones, whether multithreaded or not.

1.1 Redbook Format

The tutorial information is presented in two parts. The first section introduces some basic concepts relating to timers, queues, and events. As they are introduced, these concepts are then reinforced with examples such as:

- using timers to wait for short periods of time
- using queues to communicate between user IDs
- using event services to wait for multiple events
- using event services to replace existing unsupported event handling (for example, REXX/WAIT)

The second section focuses on using CMS′s support for multiple threads. Two areas of use are explored: first, the use of multiple threads to solve problems that might be difficult to solve in a single threaded environment; and second, the use of threads to construct servers.

Many services provided by CMS are MT aware and thread safe. Thread safe means those portions of CMS utilize a unique work area for each thread of execution. MT aware means when such services must wait, they use a mechanism that blocks only the executing thread and not the entire virtual machine.

We then describe a sample multithreaded server written in REXX that exploits two pieces of CMS that are thread safe and MT aware, REXX sockets and CPI-C support. It illustrates how quickly and easily multithreaded servers, supporting multiple concurrent users, can be constructed in REXX on CMS.

1.2 The SG245164 Package of Sample Programs

All of the programs developed during the residency and described in this redbook are available in the VM/ESA Download Library, located on the World Wide Web at URL `http://www.vm.ibm.com/download/`, in the SG245164 package. Included, as you will discover as you continue your reading of this redbook, are both a collection of relatively small, illustrative examples of CMS Application Multitasking programming techniques and a large, complete, multithreaded server application, IPGATE. All are written entirely in REXX. The multithreaded examples, including IPGATE, require the MTREXX MODULE, which you will also find in the VM/ESA Download Library in the MTREXX package.

Complete instructions for downloading packages may be found on the initial Download Library web page. A complete description of IPGATE, including the information necessary to install and configure the IPGATE server, appears in Chapter 4, “Sample Application IPGATE” on page 51.

1.3 Function Availability

All the functions documented in *CMS Application Multitasking*, SC24-5766 were added to CMS in level 9 and included in the VMMTLIB CSL library. In XA or XC mode virtual machines these CSLs were automatically loaded as part of CMS initialization. In 370 mode machines the CSLs had to be manually loaded with a `RTNLOAD` command. Normally this was done in the SYSPROF EXEC.

On CMS level 12 and later, although these functions continue to appear as RTNLOADed CSLs, they have been integrated as part of the CMS nucleus and are automatically available for use after CMS has been IPLed.

1.4 Required or Recommended Software Levels

All examples provided were developed on CMS level 14 and, except where noted, we expect them to function properly on earlier releases of CMS too. One obvious exception is that the RXSOCKET examples will only work with RXSOCKET version 3 (this is the version of RXSOCKET shipped with CMS level 13 and later) at level 3.02. Earlier levels of RXSOCKET were not MT aware nor thread safe. Neither RXSOCKET version 1 nor 2 will support multithreaded execution.

Additionally we recommend the following CMS/RXSOCKET APARs be applied:

- VM61344
- VM61477
- VM61619
- VM61645
- VM61811

Without these APARs, several of the multithreaded examples will not function correctly or at all.

If you intend to run with compiled REXX threads the following PTFs to the REXX runtime compiler are required:

- PQ12933
- PQ14488

To use CMS Pipelines in a multithreaded application, CMS level 14 (or later) is required.

Chapter 2. Enhancing Existing Applications with CMS MT Functions

The one thought we want you to come away from this chapter with is *you don't need to be a multitasking application to use CMS MT functions!* There is a tremendous amount of capability that is equally as applicable for use by traditional CMS applications as it is by multithreaded ones. In fact each example in this chapter has a filetype of EXEC. They're nothing more than standard CMS execs and each can be invoked directly from the command line!

In this chapter we'll look at three subsets of the functions that are especially useful for traditional (single threaded) CMS applications. They are:

- timer services
- queue services (interprocess communications)
- event services

2.1 Timer Services

Let's begin with a simple, yet practical, example. Often one wants an exec to wait for a short period of time. If this time period is greater than a second the CP SLEEP command may be used. For periods less than a second the following works nicely:

```
/*
 *  Delay for half a second
 */

/*
 *  Load binding files
 */
/*  call apiload 'VMREXMTR'*/
/*  call apiload 'VMREXPRO'*/
  vm_pro_success = 0
  ThreadDelay = 'VMTHRDL'

/*
 *  Delay for half a second
 */
  call csl 'ThreadDelay retcode reascode "500"'
  if retcode \= vm_pro_success then do
    say "Unexpected error from ThreadDelay"
    say " return code" retcode "reason code" reascode
  end
exit retcode
```

Figure 1. Using ThreadDelay to Delay Execution

Here we've made indirect use of timer services to suspend our execution for the requested amount of time. Once we've introduced event services we'll more fully explore timer services.

The question you probably have is *Why does this work when the exec is clearly not a thread?* The surprise here is that the exec, while not a thread itself, is running on a thread. In some regard, after release 9, CMS became a

multithreaded operating system. The traditional CMS command interpreter is one of several threads CMS always has running. For compatibility reasons the CMS command interpreter remained single threaded and does not run execs or modules on their own thread. However, because the CMS command interpreter is running on a thread, thread oriented primitives work.

2.2 Queue Services

Queue services, also known as interprocess communication, provide a conduit for data between two end points. These end points may reside within the same user ID, but they may also reside in separate user IDs or even on separate systems!

Queues whose existence is made known outside of the creating virtual machine are called *network level* queues. Network level queues allow two virtual machines to exchange messages. For simplicity, our examples here will assume the two virtual machines reside on the same system, but, given the proper network connectivity between systems, that does not have to be the case. We'll look more closely at using queues to communicate between threads within a single virtual machine when we discuss multithreaded applications.

2.2.1 CMS Requirements for Network Level Queues

Unfortunately, before we can provide a working example of a network level queue, we need to cover some minor CMS networking details. As stated above a queue has two ends. One end is established when the queue owner creates it with a `QueueCreate` call. The other end is established when a user ID opens the queue with a `QueueOpen` call.
Before a network level queue can be created two things must be done to the user ID that will create the queue:

- the user ID's directory entry must have CP IUCV authorization. This is most easily done with the addition of an `IUCV ALLOW` statement.
- the $SERVER$ NAMES file must have an entry for VMIPC. A sample $SERVER$ NAMES file suitable for most needs is:

```
:nick.VMIPC
  :list.*
```

Before a network level queue can be opened, an entry in the SCOMDIR or UCOMDIR NAMES file of the user IDs wishing to open the queue *may* be required. The reason we say *may* is that it depends on the level of CMS in use and on the name the queue creator chose for the queue. If your CMS level is older than release 12 then you will require a COMDIR entry in all cases. A sample UCOMDIR NAMES entry looks like:

```
:nick.network_level_queue_name
  :luname.*USERID userid_of_queue_creator
  :security.SAME
  :tpn.VMIPC
```

If you are running CMS release 12 or newer you may or may not require a COMDIR entry, depending on the name the queue's creator chose for the queue. On CMS release 12 and newer if, when a queue is being opened, a matching COMDIR entry is not found then, for network level queues, CMS will attempt to use the first eight characters of the queue's name to determine the user ID that owns the queue. If the first eight characters of the queue's name match the user ID of the queue creator, the queue is successfully opened. If they do not match a

valid user ID, or they do but that user ID has not created the named network level queue, then a COMDIR entry is required before CMS will be able to successfully locate and open the queue. There is no requirement however, that user IDs must be eight characters in length to create a queue that contains its name in the first eight characters. The characters that may be used in a queue name are unrestricted. This means a queue's name can even include embedded blanks. So, even short user IDs can name their queues in this manner. For example a queue name of:

```
queue_name = 'FRED     RequestQueue'
```

refers to a queue owned by user ID FRED.

Lastly, before a network level queue can be created or opened CMS will search for a matching entry in $QUEUES$ NAMES. This does not mean you require a $QUEUES$ NAMES file however, *CMS Application Multitasking* says this:
When CMS searches $QUEUES$ NAMES for an entry but does not find one, it proceeds with processing as if it had found the entry with all defaults applied. Thus, for many cases, it is not necessary to create $QUEUES$ NAMES at all.
Thankfully very reasonable defaults apply so it is unlikely you'll ever need a $QUEUES$ NAMES file. None of the queue examples in this book require a $QUEUES$ NAMES file.

Taken all together then, you can see that on recent CMS releases using queues to communicate between two local user IDs can be very simple. The queue creator requires CP IUCV authorization and an entry in $SERVER$ NAMES for VMIPC. Assuming the queue is named with the creating user ID as the first eight characters, nothing else is required before the queue can be seen and opened by any other user ID.

2.2.2 Network Level Queue Examples

Now, let's start with a simple example. A service machine that only collects data from users and does not respond to any messages. This type of data collection might be used in certain applications to track usage patterns through the application. It might also be used to track resource consumption for applications from different departments. Typically this might have been implemented by running WAKEUP in the data collection machine and using SMSG to transfer the data. This scenario may also be implemented using queues.

As we worked through the CMS networking details you saw that a queue must be created and opened in order to be used. Creating and opening a queue provide a program with a *queue handle*. The queue handle is used by other queue functions to identify the queue on which they are to operate. To send and receive data on a queue we'll now introduce the QueueSend and QueueReceiveBlock functions. Data is placed onto a queue with QueueSend. QueueReceiveBlock causes the program to block (wait) for some data to arrive on a queue. We now know enough to construct a simple data collection server.

```
  Server VM                          User VM

    QueueCreate a network              QueueOpen server's queue
       level queue                     QueueSend data
    do forever                         QueueClose server's queue
      QueueReceiveBlock to           exit
         wait for data
      Record the data
    end
    QueueDelete network
       level queue
  exit
```

Figure 2. Network Level Queues, Data Collection Overview

```
/*
 * Network level queue example, data collection, server side
 */

  parse source . . my_name .
  true  = (1=1)
  false = \true

/*
 *  Load the binding files
 */
  call apiload 'VMREXIPC'
  call apiload 'VMREXMTR'
  call apiload 'VMREXPRO'
```

Figure 3 (Part 1 of 4). Data Collection, Server Code

```
/*
 *  Create the network level queue for data collection
 */
  queue_name = left(userid(),8)||'CollectQ'
  queue_name_length = length(queue_name)
  export_level = vm_ipc_nlevel
  call csl 'QueueCreate retcode reascode queue_name',
           'queue_name_length export_level queue_handle'
  if retcode \= vm_ipc_success then do
    say my_name "Unexpected error from QueueCreate"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end

  match_key = '*'
  match_key_length = length(match_key)
  maximum_length = 4096
  timeout = 0
```

Figure 3 (Part 2 of 4). Data Collection, Server Code

```
do forever
  /*
   *  Block on the queue waiting for incoming messages
   */
    call csl 'QueueReceiveBlock retcode reascode queue_handle',
             'match_key match_key_length timeout',
             'message maximum_length returned_length',
             'key_offset key_length sender_UID sender_PID',
             'reply_token'
    if retcode \= vm_ipc_success then do
      say my_name "Unexpected error from QueueReceiveBlock"
      say my_name " return code" retcode "reason code" reascode
      exit retcode
    end
/*  call csl 'ThreadYield retcode reascode "0"' /* see VM61477 */ */

    message = left(message,returned_length)
    'EXECIO 1 DISKW' userid() 'COLLECT A ( FINIS VAR MESSAGE'
end
```

Figure 3 (Part 3 of 4). Data Collection, Server Code

```
/*
 *  Delete the queue (this code never executed)
 */
  call csl 'QueueDelete retcode reascode queue_name',
           'queue_name_length export_level'
  if retcode \= vm_ipc_success then
    select
      when reascode = vm_ipc_msgs_discarded then
        nop
      otherwise
        say my_name "Unexpected error from QueueDelete"
        say my_name " return code" retcode "reason code" reascode
        message = "ERROR Unable to shutdown gracefully,",
                  "please FORCE me"
    end
exit
```

Figure 3 (Part 4 of 4). Data Collection, Server Code

```
/*
 * Network level queue example, data collection, client side
 * You *must* change the user ID of the queue owner
 */

  address command
  signal on novalue

/*
 *  Load the binding files
 */
  call apiload 'VMREXIPC'
  call apiload 'VMREXMTR'
```

Figure 4 (Part 1 of 4). Data Collection, Client Code

```
/*
 *  Open the data queue
 */
  queue_owner = left('SRRES6',8)  /* user ID of queue owner */
  queue_name = queue_owner||'CollectQ'
  queue_name_length = length(queue_name)
  search_sequence.1 = vm_ipc_nlevel
  search_sequence.2 = -1     /* placate CSL */
  search_sequence.3 = -1     /* placate CSL */
  search_sequence_length = 1
  call csl 'QueueOpen retcode reascode queue_name',
           'queue_name_length search_sequence',
           'search_sequence_length queue_handle',
           'export_level'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueOpen"
    say " return code" retcode "reason code" reascode
    return retcode
  end
```

Figure 4 (Part 2 of 4). Data Collection, Client Code

```
/*
 *  Send the data
 */
  message = storage(0,4096)
  message_length = length(message)
  key_length = 0
  key_offset = 0
  call csl 'QueueSend retcode reascode queue_handle',
           'message message_length key_offset key_length'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueSend"
    say " return code" retcode "reason code" reascode
    return retcode
  end
```

Figure 4 (Part 3 of 4). Data Collection, Client Code

```
/*
 *  Close the data queue
 */
  call csl 'QueueClose retcode reascode queue_handle'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueClose"
    say " return code" retcode "reason code" reascode
  end

exit
```

Figure 4 (Part 4 of 4). Data Collection, Client Code

Unless the amount of data to be collected on each transfer is large (say, more than a 100 bytes), the use of queues in the above example is not demonstrably better than other methods.

An example more typical of the way CMS applications are structured would involve two way communications between a client and a server. A client would send a request to a server. Once the server had processed the request it would reply back to the client. Queues support this with the concept of replies. In place of the QueueSend function used in the last example we'll use QueueSendReply and QueueReply functions. QueueSendReply does a QueueSend and indicates that a reply is expected. When QueueSendReply is used a *reply token* is provided along with the data to QueueReceiveBlock. QueueReply performs a QueueSend like function however, rather than using a queue handle, the reply token is used to identify the queue onto which the reply should be placed. There are two benefits of QueueSendReply over QueueSend. First the reply queue does not need to be a network level queue. Second the reply queue does not need to be opened. With this we now know enough to construct a simple request/reply client/sever application.

```
Server VM                          User VM

  QueueCreate a network              QueueCreate reply queue
     level queue                     QueueOpen server's queue
  do forever                         QueueSendReply request
    QueueReceiveBlock to             QueueReceiveBlock to wait
       wait for request                 for reply
    process request                  QueueClose server's queue
    QueueReply response              QueueDelete reply queue
  end                              exit
  QueueDelete network
     level queue
exit
```

Figure 5. Network Level Queues, Client/Server Overview

```
/*
 * Network level queue example, client/server example, server side
 */

  parse source . . my_name .
  true  = (1=1)
  false = \true

/*
 *  Load the binding files
 */
  call apiload 'VMREXIPC'
  call apiload 'VMREXMTR'
  call apiload 'VMREXPRO'
```

Figure 6 (Part 1 of 4). Client/Server, Server Code

```
/*
 *  Create the network level request queue
 */
  queue_name = left(userid(),8)||'RequestQ'
  queue_name_length = length(queue_name)
  export_level = vm_ipc_nlevel
  call csl 'QueueCreate retcode reascode queue_name',
           'queue_name_length export_level queue_handle'
  if retcode \= vm_ipc_success then do
    say my_name "Unexpected error from QueueCreate"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 6 (Part 2 of 4). Client/Server, Server Code

```
  do forever
  /*
   *  Block on the queue waiting for work to arrive
   */
    match_key = '*'
    match_key_length = length(match_key)
    maximum_length = 4096
    timeout = 0
    call csl 'QueueReceiveBlock retcode reascode queue_handle',
             'match_key match_key_length timeout',
             'message maximum_length returned_length',
             'key_offset key_length sender_UID sender_PID',
             'reply_token'
    if retcode \= vm_ipc_success then do
      say my_name "Unexpected error from QueueReceiveBlock"
      say my_name " return code" retcode "reason code" reascode
      exit retcode
    end
/*  call csl 'ThreadYield retcode reascode "0"' /* see VM61477 */ */

    if reply_token <> 0 then do /* reply requested? */
      message = userid()"'s date is" date() time()
      message_length = length(message)
      key_offset = 0
      key_length = 0
      call csl 'QueueReply retcode reascode reply_token',
               'message message_length',
               'key_offset key_length'
      if retcode \= vm_ipc_success then do
        say my_name "Unexpected error from QueueReply"
        say my_name " return code" retcode "reason code" reascode
      end
    end
  end
```

Figure 6 (Part 3 of 4). Client/Server, Server Code

```
/*
 *  Delete the queue (this code never executed)
 */
  call csl 'QueueDelete retcode reascode queue_name',
           'queue_name_length export_level'
  if retcode \= vm_ipc_success then
    select
      when reascode = vm_ipc_msgs_discarded then
        nop
      otherwise
        say my_name "Unexpected error from QueueDelete"
        say my_name " return code" retcode "reason code" reascode
        message = "ERROR Unable to shutdown gracefully,",
                  "please FORCE me"
    end
exit
```

Figure 6 (Part 4 of 4). Client/Server, Server Code

```
/*
 * Network level queue example, client/server example, client side
 * You *must* change the user ID of the queue owner
 */

  address command
  signal on novalue

/*
 *  Load the binding files
 */
  call apiload 'VMREXIPC'
  call apiload 'VMREXMTR'
```

Figure 7 (Part 1 of 7). Client/Server, Client Code

```
/*
 *  Create the reply queue
 *  Note: it is *not* at network level, and hence it's
 *        name need not start with our user ID
 */
  reply_queue_name = 'ReplyQueue'
  reply_queue_name_length = length(reply_queue_name)
  export_level = vm_ipc_plevel
  call csl 'QueueCreate retcode reascode reply_queue_name',
           'reply_queue_name_length export_level reply_queue_handle'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueCreate"
    say " return code" retcode "reason code" reascode
    return retcode
  end
```

Figure 7 (Part 2 of 7). Client/Server, Client Code

```
/*
 *  Open the request queue
 */
  request_queue_owner = left('SRRES6',8)  /* user ID of queue owner */
  request_queue_name = request_queue_owner||'RequestQ'
  request_queue_name_length = length(request_queue_name)
  search_sequence.1 = vm_ipc_nlevel
  search_sequence.2 = -1     /* placate CSL */
  search_sequence.3 = -1     /* placate CSL */
  search_sequence_length = 1
  call csl 'QueueOpen retcode reascode request_queue_name',
           'request_queue_name_length search_sequence',
           'search_sequence_length request_queue_handle',
           'export_level'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueOpen"
    say " return code" retcode "reason code" reascode
    return retcode
  end
```

Figure 7 (Part 3 of 7). Client/Server, Client Code

```
/*
 *  Send the request indicating we want a reply
 */
  key_length = 0
  key_offset = 0
  message = '*'
  message_length = length(message)
  call csl 'QueueSendReply retcode reascode request_queue_handle',
           'message message_length key_offset key_length',
           'reply_queue_handle'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueSendReply"
    say " return code" retcode "reason code" reascode
    return retcode
  end
```

Figure 7 (Part 4 of 7). Client/Server, Client Code

```
/*
 *  Wait for the reply
 */
  match_key = "*"
  match_key_length = length(match_key)
  maximum_length = 1024
  timeout = 9
  call csl 'QueueReceiveBlock retcode reascode reply_queue_handle',
           'match_key match_key_length timeout message',
           'maximum_length returned_length key_offset key_length',
           'sender_UID sender_PID reply_token'
  if retcode \= vm_ipc_success then do
    select
      when reascode = vm_ipc_buf_too_small then
        nop /* for now */
      when reascode = vm_ipc_timeout then
        say "Request timed out"
      otherwise
        say "Unexpected error from QueueReceiveBlock"
        say " return code" retcode "reason code" reascode
    end
    exit
  end

  message = left(message,returned_length)
  say userid() message
```

Figure 7 (Part 5 of 7). Client/Server, Client Code

```
/*
 *  Close the request queue
 */
  call csl 'QueueClose retcode reascode request_queue_handle'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueClose"
    say " return code" retcode "reason code" reascode
  end
```

Figure 7 (Part 6 of 7). Client/Server, Client Code

```
/*
 *  Delete the reply queue
 */
  call csl 'QueueDelete retcode reascode reply_queue_name',
           'reply_queue_name_length vm_ipc_plevel'
  if retcode \= vm_ipc_success then do
    say "Unexpected error from QueueDelete"
    say " return code" retcode "reason code" reascode
  end

exit
```

Figure 7 (Part 7 of 7). Client/Server, Client Code

We've really just covered the basics needed to get network level queues working. A couple of areas you'll want to investigate further are:

- each message placed onto a queue may have a key associated with it. The receiver may also specify a key and only receive messages whose keys match.
- the list tag in $SERVER$ NAMES may be used to define a subset of user IDs that are permitted to open the network level queues created. In the sample $SERVER$ NAMES file provided earlier, the list tag was `:list.*`. An asterisk means any user ID is allowed to QueueOpen the network level queues created. If the list tag provided a list of user IDs only those user IDs are permitted to open the network level queues created. Attempts to open the queue by user IDs not in the list are rejected.

Note:

VM61722 may be required on CMS releases prior to CMS 14 before this functions properly in all cases.

Note:

You should also be aware that without APAR VM61477 some queue operations may appear to hang waiting for the partner virtual machine. Ideally the PTF should be applied, but if that is not possible, then the addition of ThreadYield calls (as shown in the examples) will bypass the problem.

As you can see queues are a very useful and powerful facility. Wouldn't it be nice if every product on VM supported a queue based command interface? Think how easy, for example, RSCS or DIRMAINT would be to work with if a queue based interface was supported as well as an SMSG/MSG based one.

2.3 Event Services

Events represent activities that occur within or to a virtual machine that may be of interest to currently executing programs. Event services are just what the name implies, services that allow a program to express an interest in one or more of these events, and be notified when they occur.

CMS provides a comprehensive set of built in events, allowing notification when a timer expires, the timezone changes, a program ABENDs or when a line is available to be read from the console, to list only a few. They are documented in *CMS Application Multitasking*; SC24-5766 and summarized in Appendix A, "Supplementary Information on System Defined Events" on page 95. In addition to the system events defined by CMS, an application may also define its own events. CMS provides identical management for system and user defined events.

2.3.1 Event Services Example

To make use of the system defined events one only needs to be familiar with a few routines. An application expresses an interest in an event, or a collection of events, with an `EventMonitorCreate` call. Just as opening a queue returned a handle that was used to identify the queue on subsequent calls, EventMonitorCreate returns a *monitor token* that we will supply on calls to other routines to identify the monitor of interest.

An application can wait for the occurrence of the event(s) defined by a monitor with an `EventWait` call.

When the application has completed its handling of the event(s) CMS may be notified by an EventMonitorReset. Often EventMonitorReset is not required since CMS will automatically ready the monitor for the next event occurrence when it can determine that the application has completed processing the current event (for example the application again enters EventWait).

Finally an EventMonitorDelete tells CMS we're no longer interested in monitoring the requested event(s). Since, in the following examples, the monitor is only used once before deletion, we could have specified vm_evn_auto_delete on the EventMonitorCreate call and had the monitor automatically deleted as part of the EventMonitorReset call. For clarity however, the monitor is explicitly deleted.

Knowing only these routines we can already make effective use of events. A simple example to illustrate their use is to wait for some console input.

```
   EventMonitorCreate to express interest
     in console events
   EventWait for the event to occur
   EventMonitorReset to finish event handling
   EventMonitorDelete to clean up event monitor
```

Figure 8. Wait for Console Input using Event Services Overview

```
/*
 *   Use event services to wait for console input
 */

  parse source . . my_name .
  true  = (1=1)
  false = \true

/*
 *  Load the binding files
 */
  call apiload 'VMREXEVN'
  call apiload 'VMREXMTR'
```

Figure 9 (Part 1 of 4). Wait for Console Input using Event Services Code

```
/*
 *  Create a monitor for unsolicited console input
 */
  monitor_flag.1 = vm_evn_no_auto_delete     /* placate CSL */
  monitor_flag.2 = vm_evn_async_monitor      /* placate CSL */
  monitor_flag.3 = vm_evn_bind_loose_signals /* placate CSL */
  monitor_flag_size = 3
  number_of_events = 1
  event_name_address.1 = 'VMCONINPUT'
  event_name_length.1 = length(event_name_address.1)
  event_key_address.1 = '*'
  event_key_length.1 = length(event_key_address.1)
  bound_signal_limit.1 = -1
  event_count = 1
  call csl 'EventMonitorCreate retcode reascode monitor_token',
           'monitor_flag monitor_flag_size number_of_events',
           'event_name_address event_name_length',
           'event_key_address event_key_length bound_signal_limit',
           'event_count'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorCreate"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 9 (Part 2 of 4). Wait for Console Input using Event Services Code

```
/*
 *  Block on the console, waiting for something to be entered
 */
  say "Please enter something"
  number_of_events = 1
  call csl 'EventWait retcode reascode monitor_token',
           'number_of_events event_flag'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventWait"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end

  parse external console_command
  say "You entered:" console_command
```

Figure 9 (Part 3 of 4). Wait for Console Input using Event Services Code

```
/*
 *  Lastly reset and delete the EventMonitor
 */
  call csl 'EventMonitorReset retcode reascode monitor_token'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorReset"
    say my_name " return code" retcode "reason code" reascode
  end
  call csl 'EventMonitorDelete retcode reascode monitor_token'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorDelete"
    say my_name " return code" retcode "reason code" reascode
  end
exit
```

Figure 9 (Part 4 of 4). Wait for Console Input using Event Services Code

2.4 Timer Services (revisited)

CMS provides a selection of routines that support timer related operations. It is possible to set a timer to expire at a specific time, or after some period of time. These periodic timers may, optionally, also be defined to be cyclical timers, that is a timer that continues to expire after each specified period. Additionally, timer support makes use of event services by signaling the VMTIMER event whenever a timer expires.

Now that you are familiar with the basics of event services you can see that it is a simple manner to combine them with CMS MT's timer support. Indeed, combining timers and events allows us to easily add timeout capability to a number of scenarios where formerly our application would have been stuck waiting forever.

Two such uses immediately spring to mind. First, continue execution, perhaps with a default value, if a prompt has not been responded to in a fixed period of time. Second, continue execution, and perhaps take corrective action, if a response to an APPC/CPIC request is not received in a timely fashion. Since the first can be easily presented as a self contained example, we present it here. However, the concept is identical for either, only the event name changes. For a CPIC example, that makes use of a variant of this technique, see the sample multitasking program, ACCCHECK, presented in Figure 17 on page 32.

```
   TimerStartInt to start timeout clock
   EventMonitorCreate to express interest
     both timer and console events
   EventWait for one of the two events to
     occur
   determine which event occurred
   TimerStopInt to clean up potential residual timer
   EventMonitorReset to finish event handling
   EventMonitorDelete to clean up event monitor
```

Figure 10. Event and Timer Combined Use Overview

```
/*
 *   Allow the user 15 seconds to input something
 */

  parse source . . my_name .
  true  = (1=1)
  false = \true

/*
 *  Load the binding files
 */
  call apiload 'VMREXEVN'
  call apiload 'VMREXMTR'
  call apiload 'VMREXTMR'
```

Figure 11 (Part 1 of 7). Event and Timer Combined Use Code

```
/*
 *  Start the timer
 */
  allowedseconds = 15
  timer_type = vm_tmr_timertype_real
  timer_cycle = vm_tmr_cycle_single
  timer_interval_units = vm_tmr_intunit_milli
  timer_interval = 1000*allowedseconds
  userword = '0000000000000000'x   /* no user word */
  call csl 'TimerStartInt retcode reascode timer_token',
           'timer_type timer_cycle timer_interval_units',
           'timer_interval userword'
  if retcode \= vm_tmr_success then do
    say my_name "Unexpected error from TimerStartInt"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 11 (Part 2 of 7). Event and Timer Combined Use Code

```
/*
 *  Create a monitor for unsolicited console input and timer
 */
  monitor_flag.1 = vm_evn_no_auto_delete     /* placate CSL */
  monitor_flag.2 = vm_evn_async_monitor      /* placate CSL */
  monitor_flag.3 = vm_evn_bind_loose_signals /* placate CSL */
  monitor_flag_size = 3
  number_of_events = 2
  event_name_address.1 = 'VMCONINPUT'
  event_name_length.1 = length(event_name_address.1)
  event_key_address.1 = '*'
  event_key_length.1 = length(event_key_address.1)
  bound_signal_limit.1 = -1
  event_name_address.2 = 'VMTIMER'
  event_name_length.2 = length(event_name_address.2)
  event_key_address.2 = d2c(timer_token,4)'E'
  event_key_length.2 = length(event_key_address.2)
  bound_signal_limit.2 = -1
  event_count = 1
  call csl 'EventMonitorCreate retcode reascode monitor_token',
           'monitor_flag monitor_flag_size number_of_events',
           'event_name_address event_name_length',
           'event_key_address event_key_length bound_signal_limit',
           'event_count'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorCreate"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 11 (Part 3 of 7). Event and Timer Combined Use Code

```
/*
 *  Block on the console/timer, waiting for something to be entered
 */
  say "You have" allowedseconds "seconds to input something"
  number_of_events = 2
  call csl 'EventWait retcode reascode monitor_token',
           'number_of_events event_flag'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventWait"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 11 (Part 4 of 7). Event and Timer Combined Use Code

```
/*
 *  Find out what event woke us up
 */
  if event_flag.1 >= 0 then do  /* keyboard input */
    parse external console_command
    say "You entered:" console_command
  end
  if event_flag.2 >= 0 then do  /* timed out */
    say "Nothing was input"
  end
```

Figure 11 (Part 5 of 7). Event and Timer Combined Use Code

```
/*
 *  Clean up the timer
 */
  call csl 'TimerStop retcode reascode timer_token intervalunits',
           'interval userword'
  if retcode \= vm_tmr_success then
    if reascode \= vm_tmr_unrecognized_token then do
      say my_name "Unexpected error from TimerStop"
      say my_name " return code" retcode "reason code" reascode
    end
```

Figure 11 (Part 6 of 7). Event and Timer Combined Use Code

```
/*
 *  Lastly reset and delete the EventMonitor
 */
  call csl 'EventMonitorReset retcode reascode monitor_token'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorReset"
    say my_name " return code" retcode "reason code" reascode
  end
  call csl 'EventMonitorDelete retcode reascode monitor_token'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorDelete"
    say my_name " return code" retcode "reason code" reascode
  end
exit
```

Figure 11 (Part 7 of 7). Event and Timer Combined Use Code

2.5 Event Services (revisited)

Recall we mentioned there can be both system and application defined events. In general, system events are created when CMS is IPLed. An event and its characteristics are defined with an EventCreate call. The occurrence of an event is made known with an EventSignal call. As well as merely indicating the occurrence of an event EventSignal can also be used to pass data associated with the event. This data may be referenced by the application via an EventRetrieve call. For system defined events information about associated signal data and related key information has been summarized in Appendix A, "Supplementary Information on System Defined Events" on page 95.

2.5.1 Replacing REXX/WAIT with Event Services

REXX/WAIT was an external REXX function that provides multiple event handling capabilities for REXX. It was an IBM internal use only package that was made available with the previous version of REXX Sockets and it has no official support. To provide compatibility, REXX Sockets continues to provide support for REXX/WAIT, however, further use of REXX/WAIT is discouraged and, as REXX Sockets now supports native CMS Event Services, any existing use should be replaced with CMS Event Services.

CMS Event Services provides support for many of the events supported by REXX/WAIT, the one glaring exception is support for notification of the arrival of reader files (REXX/WAIT called this the MAIL event). The need for a VMRDR event is well understood by CMS development and it is the authors′ hope that one will be provided in the near future.

It is a simple matter to extend our previous example to also include a demonstration of using REXX Socket′s event support and, in so doing, have it also serve as an example of how one might convert away from REXX/WAIT. Although not required, we will create our own event for REXX Sockets to signal. We do this for two reasons. First, to provide an example of how an event is created and second, to illustrate that events used with REXX Sockets must be of session scope. This scope is required since, when REXX Sockets signals this event, there can be no guarantee about which process will be running when the interrupt happens, and interrupt handling always happens in the context of the interrupted process and thread. Another approach would have been to simply use the system defined VMSOCKET event.

```
REXX/WAIT                          Event Services

initialize REXX Sockets            initialize REXX Sockets
initialize REXX/WAIT               EventCreate socket event (optional)
 (eg: SetValue calls               TimerStartInt to start timeout clock
      if needed)                   EventMonitorCreate to express interest
                                     timer, console and socket events
Wait(Timer,Cons,Socket)            EventWait for one of the events to occur
determine which event occurred     determine which event occurred
                                   TimerStopInt to clean up potential residual timer
                                   EventMonitorReset to finish event handling
                                   EventMonitorDelete to clean up event monitor
                                   EventDelete socket event
```

Figure 12. Replacing REXX/WAIT with Event Services

```
/*
 *   Illustrate using Event Services to replace REXX/WAIT
 */

  parse source . . my_name .
  true  = (1=1)
  false = \true

/*
 *  Load the binding files
 */
  call apiload 'VMREXEVN'
  call apiload 'VMREXMTR'
  call apiload 'VMREXTMR'
```

Figure 13 (Part 1 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Ready REXX Sockets for use
 */
  parse value socket('Initialize',my_name,40) with rc .
  if rc \= 0 then do
    say "Unable to initialize REXX/Socket's environment"
    exit 8
  end
  parse value socket('Socket','AF_INET','STREAM','TCP') with rc s .
  if rc \= 0 then do
    say my_name "unable to allocate a socket. RC" rc
    exit 8
  end
  parse value socket('SetSockOpt',s,'Sol_Socket','So_ASCII','On'),
              with rc .
  if rc \= 0 then do
    say my_name "unable set socket's translation. RC" rc
    exit 8
  end
  parse value socket('Bind',s,'AF_INET 12345') with rc .
  if rc \= 0 then do
    say my_name "unable to bind socket. RC" rc
    exit 8
  end
  parse value socket('Listen',s,10) with rc .
  if rc \= 0 then do
    say my_name "unable to set socket to listen. RC" rc
    exit 8
  end
```

Figure 13 (Part 2 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Create an event that REXX Sockets will signal
 *  Note - we must create the event with session scope. This is
 *         important because REXX Sockets will be signaling
 *         this event from an interrupt handler, and hence,
 *         running in a different process.
 */
  event_name = my_name
  event_name_length = length(event_name)
  event_flag.1 = vm_evn_session_scope
  event_flag.2 = vm_evn_broadcast_signals  /* placate CSL */
  event_flag.3 = vm_evn_async_signals      /* placate CSL */
  event_flag_size = 3
  loose_signal_limit = 0
  signal_timeout_period = 0
  call csl 'EventCreate retcode reascode event_name',
           'event_name_length event_flag event_flag_size',
           'loose_signal_limit signal_timeout_period'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventCreate"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 13 (Part 3 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Start the timer
 */
  allowedseconds = 120
  timer_type = vm_tmr_timertype_real
  timer_cycle = vm_tmr_cycle_single
  timer_interval_units = vm_tmr_intunit_milli
  timer_interval = 1000*allowedseconds
  userword = '0000000000000000'x   /* no user word */
  call csl 'TimerStartInt retcode reascode timer_token',
           'timer_type timer_cycle timer_interval_units',
           'timer_interval userword'
  if retcode \= vm_tmr_success then do
    say my_name "Unexpected error from TimerStartInt"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 13 (Part 4 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Create a monitor for console, timer or socket events
 */
  monitor_flag.1 = vm_evn_no_auto_delete     /* placate CSL */
  monitor_flag.2 = vm_evn_async_monitor      /* placate CSL */
  monitor_flag.3 = vm_evn_bind_loose_signals /* placate CSL */
  monitor_flag_size = 3
  number_of_events = 3
  event_name_address.1 = 'VMCONINPUT'
  event_name_length.1 = length(event_name_address.1)
  event_key_address.1 = '*'
  event_key_length.1 = length(event_key_address.1)
  bound_signal_limit.1 = -1
  event_name_address.2 = 'VMTIMER'
  event_name_length.2 = length(event_name_address.2)
  event_key_address.2 = d2c(timer_token,4)'E'
  event_key_length.2 = length(event_key_address.2)
  bound_signal_limit.2 = -1
  event_name_address.3 = event_name   /* or VMSOCKET */
  event_name_length.3 = length(event_name_address.3)
  event_key_address.3 = '*'
  event_key_length.3 = length(event_key_address.3)
  bound_signal_limit.3 = -1
  event_count = 1
  call csl 'EventMonitorCreate retcode reascode monitor_token',
           'monitor_flag monitor_flag_size number_of_events',
           'event_name_address event_name_length',
           'event_key_address event_key_length bound_signal_limit',
           'event_count'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorCreate"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 13 (Part 5 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Block on console, timer or socket
 */
  parse value socket('Select','Read' s,'Signal' event_name) with rc .
  if rc \= 0 then do
    say myname "unable to activate socket select. RC" rc
    exit 8
  end

  number_of_events = 3
  call csl 'EventWait retcode reascode monitor_token',
           'number_of_events event_flag'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventWait"
    say my_name " return code" retcode "reason code" reascode
    exit retcode
  end
```

Figure 13 (Part 6 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Find what event completed
 */
  select
    when event_flag.1 >= 0 then do  /* keyboard input */
      parse external console_command
      say "You entered:" console_command
    end
    when event_flag.2 >= 0 then do  /* timed out */
      say "Nothing was input"
    end
    when event_flag.3 >= 0 then do  /* socket */
      call csl 'EventRetrieve retcode reascode monitor_token "3"',
               'data_buffer event_flag.3 event_data_length'
      if retcode \= vm_evn_success then do
        say my_name "Unexpected error from EventRetrieve"
        say my_name " return code" retcode "reason code" reascode
      end
      parse value (data_buffer) with count 'READ' readlist,
             'WRITE' writelist 'EXCEPTION' exceptlist
      /* do some useful socket work */
    end
  end
```

Figure 13 (Part 7 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Clean up the timer
 */
  call csl 'TimerStop retcode reascode timer_token intervalunits',
           'interval userword'
  if retcode \= vm_tmr_success then
    if reascode \= vm_tmr_unrecognized_token then do
      say my_name "Unexpected error from TimerStop"
      say my_name " return code" retcode "reason code" reascode
    end
```

Figure 13 (Part 8 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 *  Reset and delete the EventMonitor and event
 */
  call csl 'EventMonitorReset retcode reascode monitor_token'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorReset"
    say my_name " return code" retcode "reason code" reascode
  end
  call csl 'EventMonitorDelete retcode reascode monitor_token'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventMonitorDelete"
    say my_name " return code" retcode "reason code" reascode
  end
  call csl 'EventDelete retcode reascode event_name event_name_length'
  if retcode \= vm_evn_success then do
    say my_name "Unexpected error from EventDelete"
    say my_name " return code" retcode "reason code" reascode
  end
```

Figure 13 (Part 9 of 10). REXX Sockets, Console and Timer Event Example

```
/*
 * Lastly terminate REXX Sockets
 */
  parse value socket('Terminate') with rc .
  if rc \= 0 then do
    say my_name "error terminating REXX sockets"
    exit 8
  end
exit
```

Figure 13 (Part 10 of 10). REXX Sockets, Console and Timer Event Example

Another event group supported by REXX/WAIT but not, and unlikely to be, supported by CMS Event Services are the message events. For multithreaded applications, PIPEs may be used to provide event support for messages. See 3.5, “Message Support for Events” on page 43 for an example of how this can be done. Additionally, the above referenced example also illustrates how an EventSignal is coded.

As with Queue Services, we've only touched upon the highlights of Event Services. There are additional routines that we have not illustrated with which you should be sure to acquaint yourself. As well be sure to fully explore all the system defined events. In addition to what we have illustrated they offer a lot of interesting capability. One practical example is using the VMERROR event to monitor applications for abends and take action, either corrective or generation of an alert notification, if one occurs. This could be done by calling EventTrap and specifying a module to be run if an application abends.

The initial sentence of this chapter told you that much of the CMS MT function was very useful for non-MT applications. We hope, through this chapter, that we have illustrated that point and, by so doing, have encouraged you to recognize where CMS MT function would be useful in enhancing your existing applications and in solving new problems. You are encouraged to read on and see how multitasking can easily be used to provide not only well performing servers, but also help in solving problems for individual users.

Chapter 3. Examples of Multithreaded Use

3.1 Basic Idea Behind Multithreaded Programs

If you are setting up an application which should work with multiple similar client requests concurrently, for example a web server or data server, the use of a single threaded server would lead to an enormous amount of logic to keep track of the data flowing from the server to the client and vice versa. Therefore you should use a multithreaded server to minimize the amount of logic to the necessary minimum.

With a multithreaded server you can easily set up an application that works with the following scheme:

```
program A

   start loop
      wait for request
         if error then leave
         else start program B      program B
         go to start loop             process client request
   exit                               exit
                                    program B
                                       process client request
                                       exit
                                     program B
                                        process client request
                                        exit
                                      program B
                                         process client request
                                         exit
```

Figure 14. Multithreaded Server Scheme

In this example a new instance of program B is started for each client request in a new thread. Each thread runs independent of the other threads (at least if it is not blocking the session or the process).

In CMS there is no (documented) function to create a thread from REXX nor could a REXX exec be the main entry point for a thread. Therefore, you need an external command to create threads from REXX.

A simple way to create threads from REXX is provided by MTREXX, a program written by Perry Ruiter. It allows you to create threads by calling a REXX external function. It creates assembler threads each of which runs the REXX interpreter.

MTREXX is available from the VM Download Library on the Web at URL `http://www.vm.ibm.com/download/`.

3.2 Functions Available to Multihreaded Programs

Each thread may use thread blocking or non-blocking calls to multitasking aware functions, such as TCP/IP socket calls through RXSOCKET or calls to the CPI Communications routines.

Note:

You should be aware that calling a program, such as XEDIT, which is process blocking also blocks all other threads in your process.

It is also possible to call pipelines within a thread without blocking the other threads.

Your threads are allowed to call other programs which then, if they call MT functions, allow the other threads to become dispatched.

With multiple threads it is possible to terminate a thread that is running a multitasking aware command that is blocking that thread. For example, testing whether a shared file pool server is responding (also a remote SFS server over AVS), would usually block your user ID if the server is not reachable or hung. This would require you to recycle your user ID (reIPL CMS or logoff). With multiple threads you are able to do this test without getting hung.

Such a test works as follows:

```
Program A
   initialize CPI conversation
   start timer
   start program B                    Program B
   wait for CPI response or timer        allocate CPI conversation
   if timer -> timeout                   send data
   else validate CPI response            exit
   delete created thread (program B)
```

Figure 15. SFS Test

The actual code is shown in the figures in the following subsections.

3.2.1 ACCCHECK EXEC

```
/*********************************************************************/
address 'COMMAND'

parse source . func fn .

parse arg sym_dest timeout .

if sym_dest = '' | sym_dest = '?' then do
   say 'Correct syntax is:' fn 'resource <timeout>'
   say 'the default timeout is 10 seconds'
   rc = 100
end
else 'MTREXX' fn sym_dest timeout

select
   when rc =    0 then
      error_text = 'Check successful'
   when rc =   10 then
      error_text = 'Timeout (No SFS Server/Target System not reachable)'
   when rc =   20 then
      error_text = 'Allocation error (Unknown Resource/set receive type)'
   when rc =   30 then
      error_text = 'Allocation error (protocol)'
   when rc =   40 then
      error_text = 'Allocation error (No AVS Mapping/resource unknown/dealloc)'
   when rc =   50 then
      error_text = 'Allocation error (receive)'
   when rc =   60 then
      error_text = 'Receive data error (Not a valid SFS response)'
   when rc =   70 then
      error_text = 'CMINIT failed (resource not valid)'
   when rc =  100 then
      error_text = 'Help given'
   when rc = 8880 then
      error_text = 'Command entered at console'
   when rc = 9990 then
      error_text = 'STOP entered at console'
   otherwise
      error_text = '???'
end

if func <> 'COMMAND' then return rc error_text

if rc <> 0 then say error_text
exit rc
```

Figure 16. ACCCHECK EXEC

3.2.2 ACCCHECK MTREXX

```
/*********************************************************************/
/* args: destination_filepool timeout_in_seconds                     */
/*                                                                   */
/* returncodes:                                                      */
/*                                                                   */
/*   10 Timeout                            no CPI response           */
/*   20 Allocation error (set receive type) Set Receive Type failed  */
/*   30 Allocation error (protocol)         invalid CPI data          */
/*   40 Allocation error (mapping/dealloc)  no mapping in remote AVS */
/*                                          /partner deallocated conv*/
/*   50 Allocation error (receive)          Receive failed            */
/*   60 Receive data error                  invalid SFS data received*/
/*  100 help given                                                   */
/* 8880 command entered at console                                   */
/* 9990 STOP entered at console                                      */
/*********************************************************************/
address 'COMMAND'

parse source . . fn .

/* definitions from CMREXX COPY ***************************************/
CM_OK                          = 0    /* return_code                */
CM_BASIC_CONVERSATION          = 0    /* conversation_type          */
CM_SEND_AND_FLUSH              = 1
CM_SEND_AND_CONFIRM            = 2
CM_SEND_AND_PREP_TO_RECEIVE    = 3
CM_SEND_AND_DEALLOCATE         = 4
CM_IMMEDIATE                   = 1
CM_RECEIVE_AND_WAIT            = 0    /* receive_type               */
CM_RECEIVE_IMMEDIATE           = 1
CM_DEALLOCATE_FLUSH            = 1
CM_DEALLOCATE_CONFIRM          = 2
CM_DEALLOCATE_ABEND            = 3
/* definitions from VMREXPRO COPY *************************************/
/*-------------------------------------------------------------*/
/*     Definition for ThreadYield                              */
/*-------------------------------------------------------------*/
THREADYIELD = VMTHRYI
/*-------------------------------------------------------------*/
/*     Definition for ThreadDelete                             */
/*-------------------------------------------------------------*/
THREADDELETE = VMTHRDE
```

Figure 17 (Part 1 of 11). ACCCHECK MTREXX

```
/* definitions from VMREXTMR COPY ***********************************/
/*-------------------------------------------------------------*/
/*     TimerStartInt function definition                       */
/*-------------------------------------------------------------*/
TIMERSTARTINT = VMTMSIN
/*-------------------------------------------------------------*/
/*     TimerStop function definition                           */
/*-------------------------------------------------------------*/
TIMERSTOP = VMTMSTP
/*-------------------------------------------------------------*/
/*     Timer types                                             */
/*-------------------------------------------------------------*/
VM_TMR_TIMERTYPE_REAL = 0
VM_TMR_TIMERTYPE_CPU = 1
/*-------------------------------------------------------------*/
/*     Timer cycles                                            */
/*-------------------------------------------------------------*/
VM_TMR_CYCLE_SINGLE = 0
VM_TMR_CYCLE_CYCLICAL = 1
/*-------------------------------------------------------------*/
/*     Interval units                                          */
/*-------------------------------------------------------------*/
VM_TMR_INTUNIT_MICRO = 0
VM_TMR_INTUNIT_MILLI = 1
```

Figure 17 (Part 2 of 11). ACCCHECK MTREXX

```
/* definitions from VMREXEVN COPY ***********************************/
/*-------------------------------------------------------------*/
/*     Monitor duration                                        */
/*-------------------------------------------------------------*/
VM_EVN_NO_AUTO_DELETE = 0
VM_EVN_AUTO_DELETE = 1
/*-------------------------------------------------------------*/
/*     Monitor synchronization                                 */
/*-------------------------------------------------------------*/
VM_EVN_ASYNC_MONITOR = 10
VM_EVN_SYNC_PROCESS_MONITOR = 11
/*-------------------------------------------------------------*/
/*     Monitor loose signal handling                           */
/*-------------------------------------------------------------*/
VM_EVN_BIND_LOOSE_SIGNALS = 20
VM_EVN_IGNORE_LOOSE_SIGNALS = 21
/*-------------------------------------------------------------*/
/*     EventMonitorCreate function definition                  */
/*-------------------------------------------------------------*/
EVENTMONITORCREATE = VMEVMCR
/*-------------------------------------------------------------*/
/*     EventWait function definition                           */
/*-------------------------------------------------------------*/
EVENTWAIT = VMEVWT
/*-------------------------------------------------------------*/
/*     EventRetrieve function definition                       */
/*-------------------------------------------------------------*/
EVENTRETRIEVE = VMEVRT
/*-------------------------------------------------------------*/
/*     EventMonitorDelete function definition                  */
/*-------------------------------------------------------------*/
EVENTMONITORDELETE = VMEVMDL
/*-------------------------------------------------------------*/
/*     EventMonitorReset function definition                   */
/*-------------------------------------------------------------*/
EVENTMONITORRESET = VMEVMRE
/********************************************************************/
error = 0                                  /* initial value       */
```

Figure 17 (Part 3 of 11). ACCCHECK MTREXX

```
arg sym_dest timeout .

if sym_dest = '*SUB*' then do
/* i have called myself as a subroutine ******************************/
   target_conv = timeout

   conv_type = CM_BASIC_CONVERSATION
   address 'CPICOMM' 'CMSCT target_conv conv_type rc'

   send_type = CM_SEND_AND_PREP_TO_RECEIVE
   address 'CPICOMM' 'CMSST target_conv send_type rc'

   ret_ctl = CM_IMMEDIATE
   address 'CPICOMM' 'CMSRT target_conv ret_ctl rc'

   address 'CPICOMM' 'CMALLC target_conv rc'

   s_buffer =             x2c('020012FFC4D4E2F3D7C94040FFC080')
   s_buffer = s_buffer || x2c('0100000005000002000000000008000')
   s_buffer = s_buffer || x2c('0000000000E0000000000000000000')
   s_buffer = s_buffer || x2c('00000000000120FF06599396720000')
   s_buffer = s_buffer || x2c(' AF712A79E7DFB80040404040404040')
   s_buffer = s_buffer || x2c('4040404040404040404BC4C1C1E5E2')
   s_buffer = s_buffer || x2c(' C7F1F0F2D7F1F4C1F9F5F1F2D4F4F6')
   s_buffer = s_buffer || x2c(' D3F2F74040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('4040404040401 0C4C5C9C2D4C6C44B')
   s_buffer = s_buffer || x2c(' C6C4C1C1E5E2C7F1AF712A79EE4700')
   s_buffer = s_buffer || x2c('01400000000000000000000000000000')
   s_buffer = s_buffer || x2c('00000000000000000000000000000000')
   s_buffer = s_buffer || x2c('000000000000000000000000000000E2')
   s_buffer = s_buffer || x2c(' C4E2C9D5404040180000000000000')
   s_buffer = s_buffer || x2c('00000100D9E3D6C340404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404000006800000200000')
   s_buffer = s_buffer || x2c('00000100000000000000000000000001')
   s_buffer = s_buffer || x2c('00004000060300000000000000300')
   s_buffer = s_buffer || x2c('00000022E5C3E4C6F1F1F1F0F140E5')
   s_buffer = s_buffer || x2c('1818173640404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('40404040404040404040404040404040')
   s_buffer = s_buffer || x2c('4040')
```

Figure 17 (Part 4 of 11). ACCCHECK MTREXX

```
   send_length = c2d(left(s_buffer,2))
   address 'CPICOMM' 'CMSEND target_conv s_buffer send_length ',
                             'req_to_send rc'
   exit rc
end
```

Figure 17 (Part 5 of 11). ACCCHECK MTREXX

```
/*********************************************************************/

if sym_dest = '' | sym_dest = '?' then do
   say 'Correct syntax is: MTREXX' fn 'resource <timeout>'
   say 'the default timeout is 10 seconds'
   exit 100                                  /* no cleanup needed   */
end

if length(sym_dest) > 8 then exit 70         /* no cleanup needed   */

address 'CPICOMM' 'CMINIT target_conv sym_dest rc'

if rc <> cm_ok then exit 70                  /* no cleanup needed   */

if timeout = '' | datatype(timeout,'W') = 0 then timeout = 10
if timeout < 1 then timeout = 10             /* use default         */

timertype = vm_tmr_timertype_real            /*                     */
cycle = vm_tmr_cycle_single                  /* oneshot             */
intervalunits = vm_tmr_intunit_milli         /* use milliseconds    */
interval = timeout * 1000                    /* number of seconds   */
userword = left('TIMEOUT',8)                 /*                     */

call csl 'TimerStartInt retcode reascode timer_token',
         'timertype cycle intervalunits interval userword'
```

Figure 17 (Part 6 of 11). ACCCHECK MTREXX

```
/*********************************************************************/
monitor_flag_size = 3                          /*                     */
                                               /*                     */
monitor_flag.1 = vm_evn_no_auto_delete         /*                     */
monitor_flag.2 = vm_evn_async_monitor          /*                     */
monitor_flag.3 = vm_evn_bind_loose_signals     /*                     */
                                               /*                     */
number_of_events = 3                           /*                     */
                                               /*                     */
event_name_address.1 = 'VMCON1ECB'             /*                     */
event_name_address.2 = 'VMTIMER'               /*                     */
event_name_address.3 = 'VMCPIC'                /*                     */

event_key_address.1 = '*'
event_key_address.2 = d2c(timer_token,4)||'%'||userword||'*'
event_key_address.3 = '*'

do i = 1 to number_of_events
   event_name_length.i = length(event_name_address.i)
   event_key_length.i = length(event_key_address.i)
   bound_signal_limit.i = -1
end

event_count = 1
call csl 'EventMonitorCreate retcode reascode monitor_token',
         'monitor_flag monitor_flag_size number_of_events',
         'event_name_address event_name_length event_key_address',
         'event_key_length bound_signal_limit event_count'
/* Start myself as a new thread ***********************************/
   parse value ThreadCreate(fn '*SUB*' target_conv) with rc threadid .
```

Figure 17 (Part 7 of 11). ACCCHECK MTREXX

```
/*********************************************************************/
   call csl 'EventWait retcode reascode monitor_token number_of_events',
            'event_flag'                       /*                     */
/* VMCON1ECB popup - console input available ************************/
      if   event_flag.1 >= 0 then do           /* VMCON1ECB           */
         do until queued() = 0                 /*                     */
            parse pull cmd                     /*                     */
            select                             /*                     */
               when translate(cmd) = 'STOP' then error = 999
               otherwise do                    /*                     */
                  address 'CMS' cmd            /*                     */
                  if rc = 0 then say 'Ready;;'/*                     */
                  else say 'Ready('rc');;'     /*                     */
                  error = 888                  /*                     */
               end                             /*                     */
            end                                /*                     */
         end                                   /*                     */
      end                                      /*                     */
```

Figure 17 (Part 8 of 11). ACCCHECK MTREXX

```
/* VMTIMER expired ***************************************************/
      if   event_flag.2 >= 0 then do          /* VMTIMER             */
         call csl 'EventRetrieve',            /* routine name        */
                  'retcode',                  /* return code         */
                  'reascode',                 /* reason code         */
                  'monitor_token',            /* monitor token       */
                  '2',                        /* event index         */
                  'retdata',                  /* returned data       */
                  '32767',                    /* max to return       */
                  'retlen'                    /* length returned     */
         error = 1

      end
```

Figure 17 (Part 9 of 11). ACCCHECK MTREXX

```
/* VMCPIC arrived *****************************************************/
      if   event_flag.3 >= 0 then do          /* VMCPIC              */
/*********************************************************************/
/* Information Input                                                 */
/*                                                                   */
/* +-------------+-----------------+-------------------+             */
/* | X'00000002' | conversation_id | event_info_length |             */
/* +-------------+-----------------+-------------------+             */
/*    4 bytes       8 bytes            4 bytes                       */
/*********************************************************************/
         call csl 'EventRetrieve',            /* routine name       */
                  'retcode',                  /* return code        */
                  'reascode',                 /* reason code        */
                  'monitor_token',            /* monitor token      */
                  '3',                        /* event index        */
                  'retdata',                  /* returned data      */
                  '32767',                    /* max to return      */
                  'retlen'                    /* length returned    */
                                              /*                    */
   if left(retdata,4) = '00000002'x then do   /*                    */
/*********************************************************************/
      parse var retdata 5 conversation_ID +8 length +4 .
                                              /*                    */
      receive_type = CM_RECEIVE_AND_WAIT

      address 'CPICOMM' 'CMSRT target_conv receive_type return_code'
      if return_code <> cm_ok then do         /*                    */
         error = 2                            /*                    */
      end                                     /*                    */
      else if c2d(length) > 0 then do         /*                    */
         r_buffer = ''                        /*                    */
         requested_length = c2d(length)       /*                    */
                                              /*                    */
         address 'CPICOMM' 'CMRCV target_conv r_buffer requested_length',
                           'data_received received_length status_received',
                               'request_to_send_received return_code'
                                              /*                    */
         if return_code <> cm_ok then do      /*                    */
            error = 5                         /*                    */
         end
         else do
            r_buffer = left(r_buffer,received_length)

            if substr(r_buffer,5,6) <> 'DMS3PZ', /*valid sfs response*/
             | substr(r_buffer,225,5) <> 'SDSCA' then do
               error = 6
            end
         end
      end
      else do
         error = 4
      end
   end
```

Figure 17 (Part 10 of 11). ACCCHECK MTREXX

```
/*********************************************************************/
   else do                                  /*                       */
      error = 3                             /*                       */
   end                                      /*                       */
end                                         /*                       */
/* now do the cleanup ************************************************/
dealloc_type = CM_DEALLOCATE_FLUSH
address 'CPICOMM' 'CMSDT target_conv dealloc_type rc'

address 'CPICOMM' 'CMDEAL target_conv rc'

call csl 'TimerStop retcode reascode timer_token',
         'intervalunits interval userword'

call csl 'ThreadDelete retcode reascode threadid'

call csl 'EventMonitorReset retcode reascode monitor_token'

call csl 'EventMonitorDelete retcode reascode monitor_token'

call csl 'ThreadYield retcode reascode 0'     /* needed to allow CMS */
                                              /* to cleanup CPIC ... */
call csl 'ThreadYield retcode reascode 0'     /* see VM61645         */

exit error * 10
```

Figure 17 (Part 11 of 11). ACCCHECK MTREXX

3.3 Synchronization of Multiple Threads

One of the more complicated tasks is the synchronization of threads. If for example preprocessing is required before some threads are allowed to run, a way to hold these threads is needed. One possible way to do this is using semaphores as the following example shows:

```
program A

    create semaphore 1 with initial_value_of_semaphore = 0
    start program B in new thread
    start program B in new thread
    start program B in new thread
    preprocessing
    reinitialize semaphore 1 -> this allows all waiting
                                threads to continue

program B

    initialization
    wait on semaphore 1
    main processing
```

Figure 18. Pausing Threads with Semaphores

The following example shows how to let a maximum number of similar threads run, as it might be used to prevent an overload of your server. The following example allows the program B to run in five threads at a time. This does not prevent the creation of more than five threads, but the other threads will stay idle until less than five threads are working.

```
Program A
   create semaphore 2 with initial_value_of_semaphore = 5
   listen to requests
      start Program B in new thread



Program B
   wait on semaphore 2   (this decreases the value of the semaphore)
   do processing
   ...
   signal semaphore 2    (this increases the value of the semaphore)
   exit
```

Figure 19. Synchronizing Threads with Semaphores

On each wait on semaphore call, the value of the semaphore is decreased by one. If the value of the semaphore is thereafter 0 or higher, no wait is required and the thread simply continues processing. Otherwise the thread will wait until another thread signals the semaphore, raising its value back to 0.

3.4 Multiple Threads with Pipelines

You can run multiple pipelines in different threads at the same time.

You can, for example, use one pipeline to connect to the message system service with starmsg and run another pipeline which works with timer driven events. Both pipelines would run totally independently and can communicate with other threads, which may use functions such as CPI-C or direct RXSOCKET calls.

This way you can use the much easier communication through events. What makes it easier is that there is no need to connect each pipeline stage or filter to all the connectors with which you want to communicate. Instead you simply signal the event, to which you want to send data, by name.

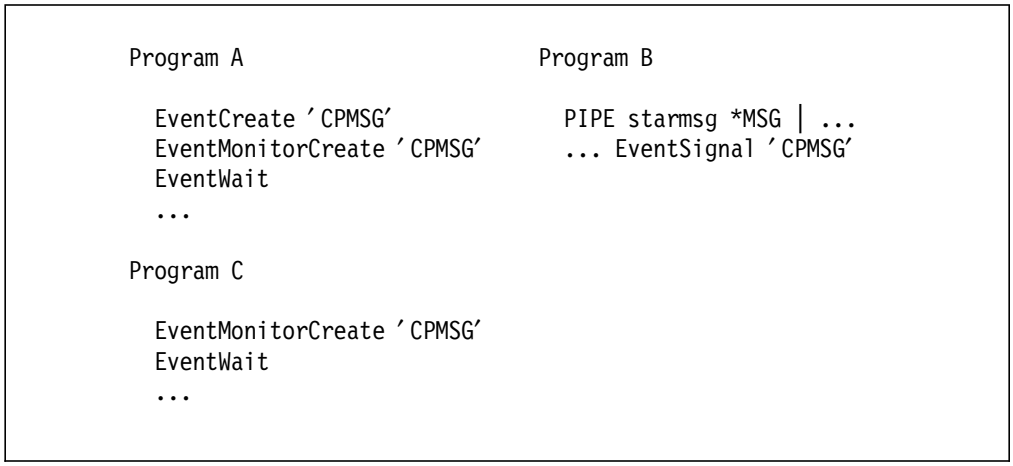


Figure 20. Signaling Messages through Pipelines

3.5 Message Support for Events

With the following small program it is possible to process messages with event signal. The messages are reformatted by a pipeline to allow an easy definition of the event keys.

The pipeline was originally part of PIPESERV (written by Finn Skovgaard).

The signaled data is in the following format:

```
....+....1....+....2....+....3....+....4....+..
msgcl   userid  nodeid  message-text...........

where msgcl  is the Message class (in hexadecimal format)
       for example:

       00000001 for local messages
       RU000001 for remote messages
       RR000001 for RSCS messages (RSCS command responses)
       00000002 for warnings
       00000003 for CPCONIO (cp command output)
       00000004 for local special messages (SMSG)
       00000005 for VMCONIO (vm command output, like say)
       00000006 for error messages (EMSG)
       00000007 for informational messages (IMSG)
       00000008 for SCIF output (secondary user)

      user ID is the originator of the message
      node ID is the node ID of the originator of the message
      message-text is the text of the message
```

Figure 21. Format of Signaled Messages

3.5.1 PIPSIGMS MTREXX

The main body of the program appears in the figure below. The code for the subroutines may be found in 4.6.8, “IPGATE Subroutines” on page 88.

```
/*********************************************************************/
                                          /*                         */
parse source . . fn ft . . env .          /*                         */
if env = 'CMS' then do                    /*                         */
                                          /*                         */
   address 'COMMAND'                      /*                         */
                                          /*                         */
   'CP SET MSG IUCV'                      /*                         */
   'CP SET SMSG IUCV'                     /*                         */
                                          /*                         */
   'PIPE command IDENTIFY | var id'       /*                         */
   parse var id thisuser +8 . 13 thisnode +8 . 26 rscsid +8 .
                                          /*                         */
   'PIPE (end ?) starmsg *MSG',           /*                         */
   '| spec 1-16 1 /'thisnode'/ 17 17-* 25',   /* Add nodeid field    */
   '| rscsmsgo: nlocate anycase 8.22 /1'rscsid||thisnode||'FROM /',
   '| rscsmsgi:faninany',                 /*Merge remote messages*/
   '| rexx ('fn ft')',                    /*                         */
   '? rscsmsgo:',                         /* Remote messages here*/
   '| rmt1: locate substr w1 of 30-* /(/',  /* What type of RSCS msg?*/
   '| xlate substr w1 of 30-* ( 40 ) 40',     /*Blank out parentheses*/
   '| spec /RU000001/ 1',                  /* Remote User prefix  */
      'substr w2 of 30-* 9',              /* Origin userid       */
      'substr w1 of 30-* 17',             /* Origin nodeid       */
      'substr w4-* of 30-* 25',           /* Text                */
   '| rscsmsgi:',                      /* Merge with iucv records */
   '? rmt1:',                    /* This is from remote RSCS server */
   '| spec /RR000001/ 1',                 /* Remote Rscs prefix  */
      'fs : substr f1 of 30-* 17',        /* Origin nodeid       */
           'substr w2-* of 30-* 25',      /* Text                */
   '| rscsmsgi:'                          /*                         */
                                          /*                         */
   'CP SET MSG ON'                        /*                         */
   'CP SET SMSG OFF'                      /*                         */
                                          /*                         */
end                                       /*                         */

/* I'm running the pipeline stage now ********************************/
else do                                   /*                         */
                                          /*                         */
   call MT_Init                           /*                         */
                                          /*                         */
   do forever                             /*                         */
      'peekto data'                       /*                         */
      if rc <> 0 then leave               /*                         */
                                          /*                         */
      call EventSignal 'CPMSG' c2x(data)  /*                         */
                                          /*                         */
      'readto'                            /*                         */
      if rc <> 0 then leave               /*                         */
   end                                    /*                         */
                                          /*                         */
   exit rc * (rc <> 12)                   /*                         */
end                                       /*                         */
exit                                      /*                         */
```

Figure 22. PIPSIGMS MTREXX

3.5.2 PROCMSG MTREXX

The program in the following figure shows how to set up the event keys to process messages and arriving reader files. The code for the subroutines may be found in 4.6.8, “IPGATE Subroutines” on page 88.

```
/*********************************************************************/
address 'COMMAND'                          /*                     */
                                           /*                     */
call MT_Init                               /*                     */
                                           /*                     */
call EventCreate 'CPMSG'                   /*                     */
                                           /*                     */
mon_tok = EventMonitorCreate('CPMSG(%%000001*)' ,
            'CPMSG(00000007%%%%%%%%%%%%%%%%RDR_FILE_%%%%_SENT_FROM_*)')
                                           /*                     */
finis = 0                                  /*                     */
                                           /*                     */
do until finis                             /*                     */
   call EventWait mon_tok                  /*                     */
                                           /*                     */
   if retcode <> 0 then do                 /*                     */
      say 'EventWait failed with return code' retcode 'reason code' reascode
      finis = 1                            /*                     */
   end                                     /*                     */
   else do                                 /*                     */
      if event_flag.1 >= 0 then do         /* Message arrived     */
                                           /*                     */
         call EventRetrieve mon_tok 1 32767
                                           /*                     */
         say 'Msg:' retdata
      end                                  /*                     */
      if event_flag.2 >= 0 then do         /* Reader file arrived */
                                           /*                     */
         call EventRetrieve mon_tok 2 32767
                                           /*                     */
         say 'Rdr:' retdata
      end                                  /*                     */
   end                                     /*                     */
end                                        /*                     */
                                           /*                     */
call EventMonitorReset mon_tok             /*                     */
call EventMonitorDelete mon_tok            /*                     */
                                           /*                     */
call EventDelete 'CPMSG'                   /*                     */
                                           /*                     */
exit                                       /*                     */
```

Figure 23. PROCMSG MTREXX

3.6 Communication between Threads

If you want to communicate between threads there are basically two different methods available:

- Communication through EventSignal
- Communication through queue-based interprocess communication (IPC)

3.6.1 Communication through EventSignal

The communication through EventSignal requires the knowledge of a single event name to be known (if a single thread should be addressed), or another identification scheme must be implemented if all threads are monitoring a single event (for example inserting the thread id into the data to be signaled).

```
Thread A
  EventCreate Event_A
  EventMonitorCreate Event_A        Thread C
  EventWait                           SignalEvent Event_A some_data
                                      SignalEvent Event_B other_data

Thread B
  EventCreate Event_B
  EventMonitorCreate Event_B
  EventWait
```

Figure 24. EventSignal with Different Events

```
Thread A
  EventCreate Event_X
  EventMonitorCreate Event_X key A
  EventWait
                                  Thread C
                                    SignalEvent Event_X A_some_data
                                    SignalEvent Event_X B_other_data
Thread B
  EventMonitorCreate Event_X key B
  EventWait
```

Figure 25. EventSignal with Different Event Keys

3.6.2 Communication through IPC

Messages that are delivered by queues can only be read once. Therefore, if you are using event signaling, you have to check if you can receive the message from the queue when you have been signaled. The message might already have been read by another thread, and a QueueReceiveBlock function would block your thread.

```
Thread A
  QueueCreate Q vm_ipc_slevel
  QueueSignalEvents Q vm_ipc_signal_on
  EventMonitorCreate VMIPC and VMTIMER ...
  EventWait
  QueueReceiveImmed

                                  Thread C
                                    QueueOpen Q vm_ipc_slevel
                                    QueueSend Q message
Thread B
  QueueOpen Q vm_ipc_slevel
  EventMonitorCreate VMIPC and VMTIMER ...
  EventWait
  QueueReceiveImmed
```

Figure 26. Communication through QueueReceiveImmed

If your thread is waiting only for messages arriving through IPC then it is much easier and cheaper to use the QueueReceiveBlock function.

```
Thread A
  QueueCreate Q vm_ipc_slevel
  QueueReceiveBlock
                                  Thread C
                                    QueueOpen Q vm_ipc_slevel
                                    QueueSend Q message
Thread B
  QueueOpen Q vm_ipc_slevel
  QueueReceiveBlock
```

Figure 27. Communication through QueueReceiveBlock

With this method only one thread is unblocked per message and there is no need for an additional check for message availability.

For synchronous communication between threads you can also use the QueueSendReply function to allow the sending thread to specify into which queue the answer should be placed.

3.7 Terminating Multithreaded Programs

There are multiple ways to terminate a multithreaded application. One possibility, which is not recommended, is the following:

Not recommended

```
call csl 'ThreadDelete retcode reascode "-1"'
```

We cannot recommend this because it will not clean up any resources acquired by this thread, nor will it free the exec's storage.

To clean up all the resources allocated by a thread, you should use one of the following methods.

3.7.1 Terminating with a Termination Event

You could define an additional event to your process (for example with the name SHUTDOWN) and add this event to all your EventMonitors. Your programs should react on the signaling of this event with a cleanup of the resources owned by the thread and use EXIT to return to the calling program. Whenever you want to stop your application, you would just signal this event.

```
Program X
   EventCreate SHUTDOWN
   ThreadCreate Program Y
   ThreadCreate Program Y
   ...                                  Program Y
   EventWait on EventMonitor               ...
   if SHUTDOWN do                          EventWait on EventMonitor
      cleanup events                       if SHUTDOWN do
      cleanup timers                          cleanup events
      ...                                     cleanup timers
      exit                                    ...
   end                                        exit
   ...                                     end
   if time_to_stop then                    ...
      EventSignal SHUTDOWN                 if time_to_stop then
                                              EventSignal SHUTDOWN
```

Figure 28. Terminating with a Termination Event

3.7.2 Terminating with EventMonitorDelete

With the previous method each thread has to monitor a specific event. A much easier method (which we prefer) is to check the retcode of an EventWait or ThreadDelay in each thread (which should be done anyway) and exit if a retcode not equal to zero is returned. This retcode not equal to zero can be achieved for EventWait by deleting its event monitor or for ThreadDelay by issuing TimerStopAll.

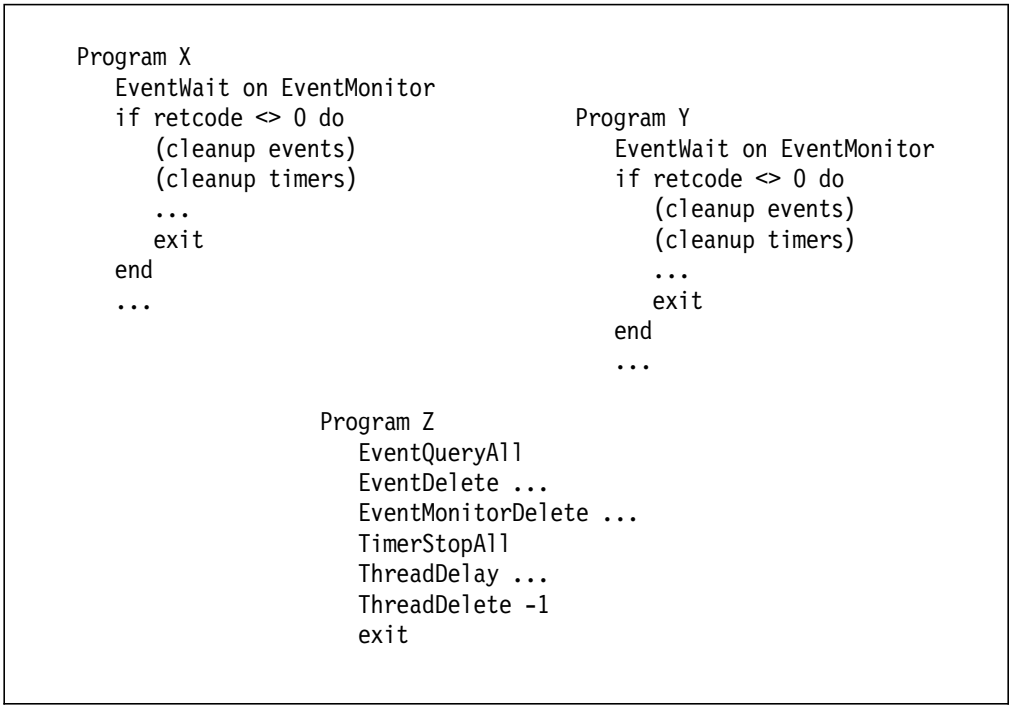


Figure 29. Terminating with EventMonitorDelete

In the above example Program Z terminates all threads in its process. This is done by deleting all EventMonitors (which are returned by EventQueryAll). The cleanup of all events can easily be done by using the output of the same call to EventQueryAll. To cleanup all timers and ThreadDelays simply call TimerStopAll. A final call to ThreadDelete may be added to dispose of any unresponsive threads.

An example of how Program Z might be coded in REXX appears in the following figure.

```
call socket 'Terminate'              /* if RXSOCKET is used */
                                     /*                     */
number_of_events   = 200
do i = 1 to number_of_events
   event_name_address.i = 200
   event_name_length.i  = 200
end
monitor_token_size = 200
                                     /*                     */
call csl 'EventQueryAll'      retcode reascode',
                             'number_of_events',
                             'event_name_address',
                             'event_name_length',
                             'actual_name_length',
                             'event_name_count',
                             'monitor_token',
                             'monitor_token_size',
                             'monitor_token_count'
                                     /*                     */
                                     /*                     */
do i = 1 to event_name_count
   event_name_address = left(event_name_address.i,actual_name_length.i)
   if left(event_name_address.i,2) <> 'VM' then do /* do not delete system events */
      say 'Event to delete:' event_name_address
      call EventDelete event_name_address
   end
end
                                     /*                     */
do i = 1 to monitor_token_count + 1
   say 'Monitor to delete:' monitor_token.i
   call EventMonitorReset monitor_token.i
   call EventMonitorDelete monitor_token.i
end
                                     /*                     */
call ThreadDelay 2000                /* allow termination   */
                                     /*                     */
call ThreadDelete '-1'               /* kill the leftover threads */

exit
```

Figure 30. Actual Code to Stop All Threads

Chapter 4. Sample Application IPGATE

4.1 What is IPGATE?

IPGATE is an application that allows you to share APPC resources across VM systems without the need of SNA.

The whole application is written in REXX and uses MTREXX to create the threads. The code might be compiled with the lastest PTFs applied to the REXX compiler and the REXX runtime library.

IPGATE communicates through TCP/IP and sets up a separate session for each conversation. One conversation is needed per user and APPC resource pair.

We tested extensively with Shared File Pools (SFS) and FCON/ESA.

Remote sessions (SNA sessions through AVS) are also supported to and from IPGATE.

4.2 Defining IPGATE to Your System

To define the IPGATE user ID to your system, the user entry in the directory should look like this:

```
USER IPGATE  IPGATE   32M 64M BG
 INCLUDE CMSUSFS
 IPL CMS PARM FILEPOOL VMSYSU AUTOCR
 IUCV *IDENT RESANY GLOBAL REVOKE
 IUCV ALLOW PRIORITY MSGLIMIT 2000
 IUCV ANY PRIORITY MSGLIMIT 2000
 OPTION QUICKDSP APPLMON MAXCONN 1000
```

Figure 31. IPGATE Sample Directory Entry

Notes:

1. The CP privilege class B is needed for the alternate user support (Diagnose X'D4'). If you have used the MODIFY DIAGNOSE facility or UCR to assign another privilege class to Diagnose X'D4', specify that class instead.
2. If you are using RACF, you should define a generic profile in VMBATCH and PERMIT IPGATE with CONTROL. If you are using VM:Secure, you should GRANT SURROGAT TO IPGATE.
3. A 191 minidisk may be used instead of an SFS filepool, if desired.
4. The IUCV statements are needed to allow IPGATE to define the APPC resources to the local system.

4.3 Configuration Files Used by IPGATE

Configuration is done with two files:

- IPGATE RESOURCE
- IPGATE USERMAP

4.3.1 IPGATE RESOURCE

The IPGATE RESOURCE file is used to identify the APPC resources that are located in the remote system and which should be made accessible to users in the local system.
The layout of this file follows:

```
*resource target   scope    target port and address
*2345678  12345678 12345678 1234 123......
VM4ALL01  VM4ALL01 SYSTEM   4567 9.165.145.192
VM4ALL02  VM4ALL02 GLOBAL   4567 9.165.145.192
VM4ALLXX  VM4ALLXX LOCAL    4567 9.165.145.192
FCXRES00  FCXRES00 SYSTEM   4567 9.165.145.192
TESTSYS   VMSYS    SYSTEM   4567 9.165.145.192
*
TESTSYSU  VMSYSU   SYSTEM   4567 9.164.155.115
```

Figure 32. IPGATE RESOURCE File Layout

An asterisk (*) in column 1 is used to identify comment lines.

resource the resource which is defined on the local system

target the resource to which the user will be connected on the remote system

scope the resource_manager_type as defined in the IDENTIFY_RESOURCE_MANAGER (XCIDRM) CPI-C routine

- **LOCAL** The resource is identified only to the system in which it resides and cannot be accessed from outside this system.
- **SYSTEM** The resource is identified only to the VM/ESA system in which it resides, but is remotely accessible from other systems.
- **GLOBAL** The resource is identified only to an entire TSAF or CS collection. It may be accessed by other users in the collection or in an SNA network.

target port and address the remote TCP/IP port and address on which the target IPGATE is listening

4.3.2 IPGATE USERMAP

The IPGATE USERMAP file has two functions. First it is used to verify that a remote system is authorized to connect to this IPGATE. To authorize a system, its IP address has to be specified in the **origin_system** columns.

The second function is the mapping of a remote user to a local user ID. This is done by looking up a matching **origin system**, **origuser** and **resource** entry.

A semicolon (;) in column one is used to identify comments. A record with an asterisk (*) in column one is a generic system entry for the user ID mapping, but is treated as a comment for purposes of verifying authorization.

The layout of this file follows:

```
;origin_system  origuser resource locuser
;23456789012345 12345678 12345678 12345678
;00.000.000.000 MAINT    VMSFS01  GUEST
9.165.145.192   HOLGER   VM4ALL01 =
9.165.145.192   HOLGER   VM4ALL02 DEMNT25
9.165.145.192   HOLGER   *        WIDMAYER
9.165.145.192   *        *        =
*               *        *        not_auth
9.164.155.115   *        *        =
```

Figure 33. IPGATE USERMAP File Layout

origin_system — the IP address of the remote systems for which resources should be made available, entered without leading zeros

origuser — the remote user ID of the accessing user

resource — the local resource name

locuser — a valid user ID defined on the local system to which the remote user ID should be mapped

4.4 Files to Install on IPGATE

The following files are necessary to run IPGATE. All are included in the sample program set for this redbook (see 1.2, “The SG245164 Package of Sample Programs” on page 2). The files can be installed on any disk or directory IPGATE accesses in its PROFILE EXEC.

IPGATE EXEC — the startup program

IPGATE1 MTREXX — the initial thread and console handler

IPGATE1L MTREXX — listens on incoming TCP requests

IPGATE1Y MTREXX — handles APPC (CPI-C) requests from user

IPGATE1I MTREXX — works with incoming TCP sessions

IPGATE1W MTREXX — monitors APPC requests for a resource

IPGATE RESOURCE — must be tailored as described above (see 4.3.1, “IPGATE RESOURCE” on page 52)

IPGATE USERMAP must be tailored as described above (see 4.3.2, “IPGATE USERMAP”)

In addition, IPGATE must have access to the following file, separately available from the VM/ESA Download Library in the MTREXX package.

MTREXX MODULE provides ThreadCreate function for REXX

4.5 Program Description of IPGATE

The following figure shows the main logic of IPGATE:

```
IPGATE EXEC
  create process MTREXX with thread IPGATE1 MTREXX

  IPGATE1 MTREXX
     create thread IPGATE1W MTREXX for each resource
     initialize RXSOCKET
     create thread IPGATE1L MTREXX to listen on a port
     wait for console input
        if stop then terminate all threads
        else execute command

     IPGATE1L MTREXX
        wait for session request over TCP/IP
           accept the session and start IPGATE1I MTREXX in a new thread

     IPGATE1W MTREXX
        identify resource
        wait for APPC request for resource
           start IPGATE1Y MTREXX in a new thread

        IPGATE1I MTREXX
           validate the authority of the requester
           find the mapping for the user ID
           connect to requested resource
           exchange data between TCP/IP session and APPC session

        IPGATE1Y MTREXX
           request a TCP/IP connection to the target IPGATE
           exchange data between TCP/IP session and APPC session
```

Figure 34. IPGATE Program Logic

4.6 The Actual IPGATE Code

In the following sections all of the code that makes up the IPGATE application is displayed for your reference. Inline subroutines have been extracted and are grouped together following the mainline routines.

4.6.1 PROFILE EXEC

```
/* */
address 'COMMAND'
if linesize() = 0 then queue 'EXEC IPGATE'
exit
```

Figure 35. Sample PROFILE EXEC of IPGATE

4.6.2 IPGATE EXEC - the Startup Program

```
/*********************************************************************/
address 'COMMAND'                          /*                     */
                                           /*                     */
'CP SET EMSG ON'                           /*                     */
                                           /*                     */
listen_port = 4567                         /*                     */
                                           /*                     */
'MTREXX IPGATE1' listen_port               /*                     */
                                           /*                     */
exit rc                                    /*                     */
```

Figure 36. IPGATE EXEC

4.6.3 IPGATE1 MTREXX - the Initial Thread and Console Handler

```
/*********************************************************************/
address 'COMMAND'                          /*                     */
                                           /*                     */
parse value '1D6O4O1DE8'x with lo hi .     /*                     */
                                           /*                     */
trace 'o'                                  /*                     */
                                           /*                     */
parse arg listen_port .                    /*                     */
                                           /*                     */
call MT_Init                               /*                     */
                                           /*                     */
'PIPE < IPGATE RESOURCE *',                /*                     */
'| nfind *'||,                             /*                     */
'| stem resource.'                         /*                     */
                                           /*                     */
'PIPE literal TCPIP DATA *',               /*                     */
'| state',                                 /*                     */
'| getfiles',                              /*                     */
'| find TCPIPUSERID '||,                   /*                     */
'| specs w2 1',                            /*                     */
'| append literal TCPIP'||,                /*                     */
'| take 1',                                /*                     */
'| var tcp_userid'                         /*                     */
```

Figure 37 (Part 1 of 5). IPGATE1 MTREXX

```
/*********************************************************************/
/* Identify the APPC resources                                       */
/*********************************************************************/
do i = 1 to resource.0                     /*                     */
                                           /*                     */
   parse value ThreadCreate('IPGATE1W' resource.i) with rc new_thread
                                           /*                     */
   if rc = 0 then do                       /*                     */
      thread.i = new_thread                /*                     */
   end                                     /*                     */
   else do                                 /*                     */
      say 'ThreadCreate failed with rc =' rc  /*                  */
   end                                     /*                     */
end                                        /*                     */
```

Figure 37 (Part 2 of 5). IPGATE1 MTREXX

```
/*********************************************************************/
/* Initialize TCP/IP communication                                   */
/*********************************************************************/
call ThreadGetId                              /*                      */
                                              /*                      */
my_name = 'IPGATE'right(thread_ID,10,0)
                                              /*                      */
parse value socket('Terminate',userid()) with rc subtaskid .
                                              /*                      */
parse value socket('Initialize',userid(),1999,tcp_userid) with rc subtaskid ,
                  maxdesc service .
                                              /*                      */
say 'Init rc =' rc                            /*                      */
                                              /*                      */
parse value socket('Socket','AF_INET','SOCK_STREAM','IPPROTO_TCP') with rc socketid .
                                              /*                      */
say 'Socket rc =' rc                          /*                      */
                                              /*                      */
parse value socket('GetHostId') with rc tcp_host_id .
                                              /*                      */
parse value Socket('SetSockOpt',socketid,'SOL_Socket','SO_REUSEADDR','On') with rc .
                                              /*                      */
parse value socket('Bind',socketid,'AF_INET' listen_port tcp_host_id) with rc .
                                              /*                      */
parse value socket('Listen',socketid,20) with rc .
                                              /*                      */
parse value Socket('SetSockOpt',socketid,'SOL_Socket','SO_Broadcast','On') with rc .
                                              /*                      */
parse value Socket('Ioctl',socketid,'FIONBIO','On') with rc .
                                              /*                      */
con_monitor_token = EventMonitorCreate('VMCON1ECB')
                                              /*                      */
finis = 0                                     /*                      */
                                              /*                      */
call ThreadCreate 'IPGATE1L' socketid listen_port
                                              /*                      */
```

Figure 37 (Part 3 of 5). IPGATE1 MTREXX

```
do until finis                                  /*                    */
                                                /*                    */
   call EventWait con_monitor_token
                                                /*                    */
   if retcode <> 0 then finis = 1               /*                    */
                                                /*                    */
   if event_flag.1 >= 0 then do                 /* VMCON1ECB          */
                                                /*                    */
      parse pull cmd                            /*                    */
      if translate(cmd) = 'STOP' then do
         finis = 1
                                                /*                    */
         say 'Terminating ...'                  /*                    */
                                                /*                    */
         call socket 'Terminate'                /*                    */
                                                /*                    */
         number_of_events   = 200
         do i = 1 to number_of_events
            event_name_address.i = 200
            event_name_length.i  = 200
         end
         monitor_token_size = 200
                                                /*                    */
         call csl 'EventQueryAll      retcode reascode',
                                  'number_of_events',
                                  'event_name_address',
                                  'event_name_length',
                                  'actual_name_length',
                                  'event_name_count',
                                  'monitor_token',
                                  'monitor_token_size',
                                  'monitor_token_count'
                                                /*                    */
         do i = 1 to event_name_count
            event_name_address = left(event_name_address.i,actual_name_length.i)
            if left(event_name_address.i,2) <> 'VM' then do
               say 'Event to delete:' event_name_address
               call EventDelete event_name_address
            end
         end
                                                /*                    */
         do i = 1 to monitor_token_count + 1
            say 'Monitor to delete:' monitor_token.i
            call EventMonitorReset monitor_token.i
            call EventMonitorDelete monitor_token.i
         end
                                                /*                    */
         call ThreadDelay 2000                  /*                    */
                                                /*                    */
         call ThreadDelete '-1'                 /*                    */
      end                                       /*                    */
```

Figure 37 (Part 4 of 5). IPGATE1 MTREXX

```
      else do                                  /*                    */
         address 'CMS' cmd                     /*                    */
         if rc = 0 then say 'Ready;;'          /*                    */
         else say 'Ready('rc');;'              /*                    */
      end                                      /*                    */
   end                                         /*                    */
end                                           /*                    */
/*********************************************************************/
exit                                          /*                    */
```

Figure 37 (Part 5 of 5). IPGATE1 MTREXX

4.6.4 IPGATE1L MTREXX - Listens on Incoming TCP Requests

```
/*********************************************************************/
address 'COMMAND'                             /*                    */
                                              /*                    */
parse value '1D60401DE8'x with lo hi .        /*                    */
                                              /*                    */
trace 'o'                                     /*                    */
                                              /*                    */
arg socketid listen_port .                    /*                    */
                                              /*                    */
call MT_Init                                  /*                    */
                                              /*                    */
call ThreadGetId                              /*                    */
                                              /*                    */
my_name = 'IPGATEL'right(thread_ID,10,0)      /*                    */
                                              /*                    */
say my_name 'started ('socketid listen_port')'/*                    */
                                              /*                    */
call EventCreate my_name                      /*                    */
                                              /*                    */
tcp_monitor_token = EventMonitorCreate(my_name)
                                              /*                    */
parse value socket('Select','Read' socketid 'Write',
                           'Exception','SIGNAL' my_name) with rc .
                                              /*                    */
finis = 0                                     /*                    */
                                              /*                    */
```

Figure 38 (Part 1 of 3). IPGATE1L MTREXX

```
do until finis                                  /*                    */
                                                /*                    */
   call EventWait tcp_monitor_token             /*                    */
                                                /*                    */
   if retcode <> 0 then finis = 1               /*                    */
                                                /*                    */
   if event_flag.1 >= 0 then do                 /* tcp-event          */
                                                /*                    */
      call EventRetrieve tcp_monitor_token 1    /*                    */
                                                /*                    */
      parse var retdata tcp_events ' READ ' tcp_read ,
      ' WRITE ' tcp_write ' EXCEPTION ' tcp_except
                                                /*                    */
      if tcp_read <> '' then do
                                                /*                    */
         parse value socket('Accept',tcp_read) with rc new_socket
         parse var new_socket new_socket_id new_family .

         if rc = 0 then do

            parse value ThreadCreate('IPGATE1I' thread_ID ,
            new_socket ) with rc new_threadid .

         end
         else do
            say my_name||hi'Accept: rc =' rc new_socket lo
         end
      end                                       /*                    */
      if tcp_write <> '' then do
                                                /*                    */
         parse value socket('Write',tcp_write,'FFFFFFFF'x) with rc
                                                /*                    */
      end                                       /*                    */
      if finis = 0 then                         /*                    */
      parse value socket('Select','Read' socketid ,
      'Write Exception','SIGNAL' my_name) with rc .
   end                                          /*                    */
end                                             /*                    */
```

Figure 38 (Part 2 of 3). IPGATE1L MTREXX

```
/*********************************************************************/
call EventMonitorReset tcp_monitor_token        /*                    */
call EventMonitorDelete tcp_monitor_token       /*                    */
                                                /*                    */
call EventDelete my_name                        /*                    */
                                                /*                    */
exit                                            /*                    */
```

Figure 38 (Part 3 of 3). IPGATE1L MTREXX

4.6.5 IPGATE1Y MTREXX - Handles APPC (CPI-C) Requests from User

```
/*********************************************************************/
/*                                                                   */
/* Allocation Request                                                */
/*                                                                   */
/* +-------------+-----------------+                                 */
/* | X'00000001' |   resource_id   |                                 */
/* +-------------+-----------------+                                 */
/*    4 bytes       8 bytes                                          */
/*                                                                   */
/*                                                                   */
/* Information Input                                                 */
/*                                                                   */
/* +-------------+-----------------+------------------+              */
/* | X'00000002' | conversation_id | event_info_length |              */
/* +-------------+-----------------+------------------+              */
/*    4 bytes       8 bytes            4 bytes                       */
/*                                                                   */
/*                                                                   */
/* Resource revoked notification                                     */
/*                                                                   */
/* +-------------+-----------------+                                 */
/* | X'00000003' |   resource_id   |                                 */
/* +-------------+-----------------+                                 */
/*    4 bytes       8 bytes                                          */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
address 'COMMAND'                          /*                     */
                                           /*                     */
parse value '1D6O4O1DE8'x with lo hi .     /*                     */
                                           /*                     */
trace 'o'                                  /*                     */
                                           /*                     */
call MT_Init                               /*                     */
                                           /*                     */
parse arg loc_resource .                   /*                     */
                                           /*                     */
call ThreadGetId                           /*                     */
                                           /*                     */
my_name = 'IPGATEY' ||  right(thread_ID,1O,0)  /*                 */
                                           /*                     */
'PIPE (end ?) < IPGATE RESOURCE *',        /*                     */
'| nfind *'||,                             /*                     */
'| find' left(loc_resource,9,'_') ||,
'| targ: fanout',
'| specs w2 1',                           /*                     */
'| var target_resource',                  /*                     */
'? targ:',                                 /*                     */
'| specs w4-5 1',                         /*                     */
'| var target_ip'                         /*                     */
```

Figure 39 (Part 1 of 18). IPGATE1Y MTREXX

```
                                          /*                    */
if verify(word(target_ip,2),'0123456789.') > 0 then,
   do i = 1 to 5 until rc = 0
   parse value socket('Resolve',word(target_ip,2)) with rc ip_addr .
end                                       /*                    */
else ip_addr = word(target_ip,2)          /*                    */
                                          /*                    */
if rc = 0 then do                         /*                    */
   target_ip = word(target_ip,1) ip_addr  /*                    */
end                                       /*                    */
else do                                   /*                    */
   say my_name||hi'Nameserver lookup failed - rc =' rc lo
   address 'CPICOMM' 'CMACCP cm_convid cm_rc'
   dealloc_type = CM_DEALLOCATE_ABEND
   address 'CPICOMM' 'CMSDT cm_convid dealloc_type cm_rc'
   address 'CPICOMM' 'CMDEAL cm_convid cm_rc'
   exit rc                                /*                    */
end                                       /*                    */
                                          /*                    */
say my_name 'started.'                    /*                    */
```

Figure 39 (Part 2 of 18). IPGATE1Y MTREXX

```
/*********************************************************************/
/* Initialize TCP/IP communication                                   */
/*********************************************************************/
parse value socket('Socket','AF_INET','SOCK_STREAM','IPPROTO_TCP') with rc socketid .

parse value socket('Connect',socketid,'AF_INET' target_ip) with rc .

call EventCreate my_name                  /*                    */
```

Figure 39 (Part 3 of 18). IPGATE1Y MTREXX

```
/*********************************************************************/
/* Initialize APPC   communication                                   */
/*********************************************************************/
fill = CM_FILL_LL                          /*                        */
fill_buffer = CM_FILL_BUFFER               /*                        */
/*********************************************************************/
montok1 = EventMonitorCreate('VMCPIC' || '(' || '00000002'x || '*)')
                                           /*                        */
address 'CPICOMM' 'CMACCP cm_convid cm_rc'  /*                        */
                                           /*                        */
montok = EventMonitorCreate('VMCPIC' || '(' || '00000002'x || left(cm_convid,8,'_') || '*)',
                         my_name)
                                           /*                        */
/* set fill ********************************************************/
address 'CPICOMM' 'CMSF cm_convid fill cm_rc' /*                      */
if cm_return_code.cm_rc <> 'CM_OK' then ,
   address 'CPICOMM' 'CMSF cm_convid fill_buffer cm_rc'
                                           /*                        */
address 'CPICOMM' 'CMEPLN cm_convid partner_LU_name ',
   'partner_LU_name_length return_code'
work_user = word(partner_LU_name,2)        /*                        */
                                           /*                        */
if word(partner_LU_name,1) <> '*USERID' then do
/* remote access ----------------------------**********************/
   address 'CPICOMM' 'XCECSU cm_convid security_user_ID ',
      'security_user_ID_length rc'
   if cm_return_code.rc = 'CM_OK' then ,   /*                        */
      work_user = left(security_user_ID,security_user_ID_length)
                                           /*                        */
   address 'CPICOMM' 'XCELFQ cm_convid local_FQ_LU_name',
                     'local_FQ_LU_name_length return_code'
                                           /*                        */
   address 'CPICOMM' 'XCERFQ cm_convid remote_FQ_LU_name',
                     'remote_FQ_LU_name_length return_code'
                                           /*                        */
   address 'CPICOMM' 'XCETPN cm_convid TP_name',
                     'TP_name_length return_code'
end                                        /*                        */
                                           /*                        */
```

Figure 39 (Part 4 of 18). IPGATE1Y MTREXX

```
say my_name 'Request from'hi||work_user||lo'for'hi||target_resource||,
            lo'at'hi||target_ip||lo           /*                      */
                                              /*                      */
text = '*IPGATE*INIT*'target_resource'*'work_user'*'
                                              /*                      */
do until text = '' | rc <> 0                  /*                      */
                                              /*                      */
   parse value socket('Write',socketid,text) with rc bytes_sent
                                              /*                      */
   if rc <> 0 then do                         /*                      */
                                              /*                      */
      say my_name||hi'Init Write failed with rc =' rc '('bytes_sent')'lo
                                              /*                      */
      call socket 'Close',socketid            /*                      */
                                              /*                      */
      call EventDelete my_name                /*                      */
                                              /*                      */
      call EventMonitorReset montok           /*                      */
      call EventMonitorDelete montok          /*                      */
                                              /*                      */
      call EventMonitorReset montok1          /*                      */
      call EventMonitorDelete montok1         /*                      */
                                              /*                      */
      dealloc_type = CM_DEALLOCATE_CONFIRM
      address 'CPICOMM' 'CMSDT cm_convid dealloc_type cm_rc'
      address 'CPICOMM' 'CMDEAL cm_convid cm_rc'
                                              /*                      */
      dealloc_type = CM_DEALLOCATE_ABEND
      address 'CPICOMM' 'CMSDT cm_convid dealloc_type cm_rc'
      address 'CPICOMM' 'CMDEAL cm_convid cm_rc'
                                              /*                      */
      exit rc                                 /*                      */
   end                                        /*                      */
   else do
      if bytes_sent = length(text) then do
         text = ''                            /*                      */
      end                                     /*                      */
      else do                                 /*                      */
         text = substr(text,bytes_sent+1)
      end                                     /*                      */
   end                                        /*                      */
end                                           /*                      */
                                              /*                      */
```

Figure 39 (Part 5 of 18). IPGATE1Y MTREXX

```
parse value Socket('Read',socketid) with rc length data
if rc <> 0 | data <> '00'x then do            /*                     */
   say my_name||hi'Init Read failed with rc =' rc '('length data')'lo
                                              /*                     */
   call socket 'Close',socketid               /*                     */
                                              /*                     */
   call EventDelete my_name                   /*                     */
                                              /*                     */
   call EventMonitorReset montok              /*                     */
   call EventMonitorDelete montok             /*                     */
                                              /*                     */
   call EventMonitorReset montok1             /*                     */
   call EventMonitorDelete montok1            /*                     */
                                              /*                     */
   dealloc_type = CM_DEALLOCATE_ABEND
   address 'CPICOMM' 'CMSDT cm_convid dealloc_type cm_rc'
   address 'CPICOMM' 'CMDEAL cm_convid cm_rc'
                                              /*                     */
   exit rc                                    /*                     */
end                                           /*                     */
```

Figure 39 (Part 6 of 18). IPGATE1Y MTREXX

```
/* now set the socket to non-blocking mode ***************************/
parse value Socket('SetSockOpt',socketid,'SOL_Socket','SO_Broadcast','On') with rc .

parse value Socket('Ioctl',socketid,'FIONBIO','On') with rc .

sel_write = ''                                /*                     */
                                              /*                     */
parse value socket('Select','Read' socketid 'Write' sel_write ,
                   'Exception' socketid,'SIGNAL' my_name,) with rc
```

Figure 39 (Part 7 of 18). IPGATE1Y MTREXX

```
/*********************************************************************/
$REM_INFO$ = left(work_user,8) ip_addr target_resource
$LOC_INFO$ = left(work_user,8) socketid date() time()
$IP_DATA_OUT$ = 0                             /*                     */
$IP_DATA_IN$ = 0                              /*                     */
$LAST_ACTIV$ = date() time()                  /*                     */
                                              /*                     */
finis = 0                                     /*                     */
                                              /*                     */
first = 1                                     /*                     */
ev_wait_tok = montok1                         /*                     */
text = ''                                     /*                     */
tcp_text = ''                                 /*                     */
                                              /*                     */
```

Figure 39 (Part 8 of 18). IPGATE1Y MTREXX

```
do until finis                                /*                     */
                                              /*                     */
   call EventWait ev_wait_tok                 /* wait for event      */
                                              /*                     */
   if retcode <> 0 then finis = 1             /*                     */
                                              /*                     */
$LAST_ACTIV$ = date() time()                  /*                     */
```

Figure 39 (Part 9 of 18). IPGATE1Y MTREXX

```
/*-------------------------------------------*************************/
   if event_flag.1 >= 0 then do               /* VMCPIC              */
                                              /*                     */
      call EventRetrieve ev_wait_tok 1 32767  /*                     */
                                              /*                     */
      if left(retdata,4) = '00000002'x then do /* Information input */
         parse var retdata . 5 conv_id 13 length +4 .
         if conv_id <> cm_convid then iterate /*                     */
         if first then do                     /*                     */
            ev_wait_tok = montok              /*                     */
            call EventMonitorReset  montok1   /*                     */
            call EventMonitorDelete montok1   /*                     */
            first = 0                         /*                     */
         end                                  /*                     */
         total_sent = c2d(length)             /*                     */
         total_received = 0                   /*                     */
/*-------------------------------------------*************************/
         address 'CPICOMM' 'CMECS cm_convid conversation_state return_code'
                                              /*                     */
if cm_conversation_state.conversation_state <> 'CM_RECEIVE_STATE' & ,
   cm_conversation_state.conversation_state <> 'CM_SEND_PENDING_STATE' then,
   say my_name 'Rcv_state' cm_conversation_state.conversation_state
                                              /*                     */
         if total_sent = 0 then finis = 1     /*                     */
```

Figure 39 (Part 10 of 18). IPGATE1Y MTREXX

```
/*-----------------------------------------*************************/
         else if cm_conversation_state.conversation_state = 'CM_RECEIVE_STATE' ,
         then do until total_received >= total_sent
            receive_type = CM_RECEIVE_AND_WAIT/*                    */
            address 'CPICOMM' 'CMSRT cm_convid receive_type return_code'
                                            /*                      */
if return_code <> 0 then say my_name 'CMSRT' cm_return_code.return_code
                                            /*                      */
            requested_length = c2d(length)
            if requested_length > 32767 then requested_length = 32767
                                            /*                      */
            r_buffer = ''                   /*                      */
                                            /*                      */
            address 'CPICOMM' 'CMRCV cm_convid r_buffer requested_length',
                                 'data_received received_length status_received',
                                 'request_to_send_received return_code'
                                            /*                      */
if return_code <> 0 then say my_name 'CMRCV' cm_return_code.return_code
                                            /*                      */
            if substr(r_buffer,3,8) = '12FFC4D4E2F3D7C9'x then do
               r_buffer = overlay(copies('00'x,24),r_buffer,85)
               r_buffer = overlay(copies('00'x,26),r_buffer,157)
            end                             /*                      */
                                            /*                      */
            if cm_return_code.return_code = 'CM_RESOURCE_FAILURE_NO_RETRY',
             | cm_return_code.return_code = 'CM_PROGRAM_PARAMETER_CHECK' then do
               finis = 1                    /*                      */
               leave                        /*                      */
            end                             /*                      */
            total_received = total_received + received_length
                                            /*                      */
            send_type = CM_BUFFER_DATA      /*                      */
                                            /*                      */
            if total_received >= total_sent & ,
               cm_status_received.status_received <> 'CM_NO_STATUS_RECEIVED' then ,
               send_type = CM_SEND_AND_PREP_TO_RECEIVE
                                            /*                      */
            text = text'*IPGATE*CMSST*'send_type'*'
                                            /*                      */
            s_buffer = left(r_buffer,received_length)
            send_length = c2d(left(r_buffer,2))
                                            /*                      */
            text = text || '*IPGATE*CMSEND*'||length(s_buffer)||'*'||,
                   s_buffer||'*'
                                            /*                      */
```

Figure 39 (Part 11 of 18). IPGATE1Y MTREXX

```
            if sel_write = '' then do         /*                     */
               wr_text = left(text,min(50000,length(text)))
               parse value socket('Write',socketid,wr_text) with rc bytes_sent .
               if rc <> 0 & rc <> 35 then do     /*                     */
                  finis = 1                      /*                     */
                  say my_name||hi'write failed with rc =' rc '('bytes_sent')'lo
               end                               /*                     */
               if rc = 35 then do
                  if sel_write = '' then do      /*                     */
                     sel_write = socketid /* EWOULDBLOCK      */
                     parse value socket('Select','Read',
                     'Write' sel_write ,
                     'Exception','SIGNAL' my_name) with rc .
                  end                            /*                     */
               end                               /*                     */
               if rc = 0 then do
                  $IP_DATA_OUT$ = $IP_DATA_OUT$ + bytes_sent
                  if bytes_sent = length(text) then do
                     text = ''                   /*                     */
                     sel_write = ''              /*                     */
                  end                            /*                     */
                  else do                        /*                     */
                     text = substr(text,bytes_sent+1)
                     if sel_write = '' then do      /*                     */
                        sel_write = socketid /* EWOULDBLOCK      */
                        parse value socket('Select','Read',
                        'Write' sel_write ,
                        'Exception','SIGNAL' my_name) with rc .
                     end                               /*                     */
                  end                            /*                     */
               end                            /*                     */
            end                               /*                     */
         end                                  /*                     */
/*-----------------------------------------*************************/
      end                                     /*                     */
      else do                                 /*                     */
         say my_name||hi'Unexpected APPC input ('retdata '-' c2x(retdata)')',
                     'or termination request'lo
         finis = 1                            /*                     */
      end                                     /*                     */
   end                                        /*                     */
```

Figure 39 (Part 12 of 18). IPGATE1Y MTREXX

```
/*-----------------------------------------*************************/
   if event_flag.2 >= 0 then do              /* TCP/IP              */
                                             /*                     */
      ip_read_done = 0                       /*                     */
                                             /*                     */
      read_rc = 0                            /*                     */
                                             /*                     */
      call EventRetrieve ev_wait_tok 2 32767 /*                     */

      parse var retdata tcp_events ' READ ' tcp_read ' WRITE ' tcp_write ,
                        ' EXCEPTION ' tcp_except
                                             /*                     */
if (tcp_read <> socketid) & tcp_read <> '' then ,
   say my_name||hi'Read for' tcp_read 'signaled - should be' socketid||lo
if (tcp_write <> socketid) & tcp_write <> '' then ,
   say my_name||hi'Write for' tcp_write 'signaled - should be' socketid||lo
if (tcp_except <> socketid) & tcp_except <> '' then ,
   say my_name||hi'Exception for' tcp_except 'signaled - should be' socketid||lo
```

Figure 39 (Part 13 of 18). IPGATE1Y MTREXX

```
/*- receiving tcp answer ---------------------*************************/
      if tcp_read <> '' then do
         parse value socket('Read',socketid,100000) with rc bytes data
         read_rc = rc                        /*                     */
         if rc = 0 then ,                    /*                     */
         $IP_DATA_IN$ = $IP_DATA_IN$ + bytes
         if rc <> 35 & (rc <> 0 | bytes = 0) then do
            say my_name||hi'Thread terminating ... Read rc =' rc '('bytes data')'lo
            finis = 1
         end
```

Figure 39 (Part 14 of 18). IPGATE1Y MTREXX

```
         else if rc <> 35 then do until ip_read_done
            tcp_text = tcp_text || data       /*                     */
            data = ''                         /*                     */
            select                            /*                     */
/*- set send type received *IPGATE*CMSST*n* --***********************/
               when left(tcp_text,14) = '*IPGATE*CMSST*',
                  & length(tcp_text) >= 16 then do
                                              /*                     */
                  parse var tcp_text '*IPGATE*CMSST*'send_type'*'tcp_text
                  address 'CPICOMM' 'CMSST cm_convid send_type rc'
               end                            /*                     */
/*- send data received *IPGATE*CMSEND*length*data* ******************/
               when left(tcp_text,15) = '*IPGATE*CMSEND*',
                  & length(tcp_text) >= 19 then do
                                              /*                     */
                  parse var tcp_text '*IPGATE*CMSEND*'tcp_temp
                  if index(tcp_text,'*') > 0 then do
                                              /*                     */
                     parse var tcp_text '*IPGATE*CMSEND*'length'*'tcp_temp
                     if length(tcp_temp) >= length + 1 then do /* data complete ? */
                                              /*                     */
                        cm_data = left(tcp_temp,length)
                        if length(tcp_temp) = length + 1 then,
                           tcp_text = ''      /*                     */
                        else tcp_text = substr(tcp_temp,length+2)
                                              /*                     */
                        address 'CPICOMM' 'CMECS cm_convid',
                                'conversation_state return_code'
                                              /*                     */
if cm_conversation_state.conversation_state <> 'CM_SEND_STATE' & ,
   cm_conversation_state.conversation_state <> 'CM_SEND_PENDING_STATE' then,
   say my_name 'Send_state' cm_conversation_state.conversation_state
                                              /*                     */
                        address 'CPICOMM' 'CMSEND cm_convid cm_data',
                                          'length req_to_send rc'
                                              /*                     */
if rc <> 0 then say my_name 'CMSEND' cm_return_code.rc
                                              /*                     */
                     end                      /*                     */
                     else ip_read_done = 1    /*                     */
                  end                         /*                     */
                  else ip_read_done = 1       /*                     */
               end                            /*                     */
/*- else wait for complete data -------------************************/
               otherwise ip_read_done = 1     /*                     */
            end                               /*                     */
         end
      end
```

Figure 39 (Part 15 of 18). IPGATE1Y MTREXX

```
/*-------------------------------------------*************************/
/* if write is signaled, the remaining data to send must be in       */
/* variable text                                                     */
/*-------------------------------------------*************************/
      if tcp_write <> '' then do              /*                      */
         if text = '' then do                 /* nothing to write ?   */
            sel_write = ''                    /*                      */
         end                                  /*                      */
         else do                              /*                      */
            wr_text = left(text,min(50000,length(text)))
            parse value socket('Write',socketid,wr_text) with rc bytes_sent .
/*-------------------------------------------*************************/
            if rc <> 0 & rc <> 35 then do     /*                      */
               finis = 1                      /*                      */
               say my_name||hi'write failed with rc =' rc '('bytes_sent')'lo
            end                               /*                      */
            if rc = 35 then do
               if sel_write = '' then do      /*                      */
                  sel_write = socketid /* EWOULDBLOCK      */
               end                            /*                      */
            end                               /*                      */
            if rc = 0 then do
               $IP_DATA_OUT$ = $IP_DATA_OUT$ + bytes_sent
               if bytes_sent = length(text) then do
                  text = ''                   /*                      */
                  sel_write = ''              /*                      */
               end                            /*                      */
               else do                        /*                      */
                  text = substr(text,bytes_sent+1)
                  if sel_write = '' then do   /*                         */
                     sel_write = socketid /* EWOULDBLOCK      */
                  end                         /*                         */
               end                            /*                      */
            end                               /*                      */
/*-------------------------------------------*************************/
         end                                  /*                      */
      end
```

Figure 39 (Part 16 of 18). IPGATE1Y MTREXX

```
/*-------------------------------------------*************************/
      if tcp_except <> '' then do
         parse value socket('Recv',socketid,,'OUT_OF_BAND') with rc bytes data
         say my_name 'exception read' bytes 'bytes.'
         say my_name 'exception data:' c2x(data)
      end
      if finis = 0  then ,                    /*                      */
      parse value socket('Select','Read' socketid 'Write' sel_write ,
                         'Exception' socketid,'SIGNAL' my_name,) with rc
   end                                        /*                      */
end                                           /*                      */
```

Figure 39 (Part 17 of 18). IPGATE1Y MTREXX

```
/*-----------------------------------------*************************/
dealloc_type = CM_DEALLOCATE_ABEND          /*                       */
address 'CPICOMM' 'CMSDT cm_convid dealloc_type rc'
address 'CPICOMM' 'CMDEAL cm_convid rc'
                                            /*                       */
call EventMonitorReset ev_wait_tok          /*                       */
call EventMonitorDelete ev_wait_tok         /*                       */
                                            /*                       */
call socket 'Close',socketid                /*                       */
                                            /*                       */
call EventDelete my_name                    /*                       */
                                            /*                       */
say my_name 'ended.'                        /*                       */
                                            /*                       */
exit                                        /*                       */
```

Figure 39 (Part 18 of 18). IPGATE1Y MTREXX

4.6.6 IPGATE1I MTREXX - Works with Incoming TCP Sessions

```
/*********************************************************************/
/*                                                                   */
/* Allocation Request                                                */
/*                                                                   */
/* +-------------+-----------------+                                 */
/* | X'00000001' |   resource_id   |                                 */
/* +-------------+-----------------+                                 */
/*    4 bytes       8 bytes                                          */
/*                                                                   */
/*                                                                   */
/* Information Input                                                 */
/*                                                                   */
/* +-------------+-----------------+------------------+              */
/* | X'00000002' | conversation_id | event_info_length |              */
/* +-------------+-----------------+------------------+              */
/*    4 bytes       8 bytes            4 bytes                       */
/*                                                                   */
/*                                                                   */
/* Resource revoked notification                                     */
/*                                                                   */
/* +-------------+-----------------+                                 */
/* | X'00000003' |   resource_id   |                                 */
/* +-------------+-----------------+                                 */
/*    4 bytes       8 bytes                                          */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
address 'COMMAND'                              /*                   */
                                               /*                   */
parse value '1D60401DE8'x with lo hi .         /*                   */
                                               /*                   */
trace 'o'                                      /*                   */
                                               /*                   */
call MT_Init                                   /*                   */
                                               /*                   */
parse arg calling_thread socketid family rem_port rem_ip
                                               /*                   */
call ThreadGetId                               /*                   */
                                               /*                   */
my_name = 'IPGATEI' ||  right(thread_ID,10,0)  /*                   */
                                               /*                   */
say my_name 'started. ('calling_thread socketid family rem_port rem_ip')'
                                               /*                   */
```

Figure 40 (Part 1 of 16). IPGATE1I MTREXX

```
'PIPE < IPGATE USERMAP',                        /*                    */
'| nfind ;'||,                                  /*                    */
'| nfind *'||,                                  /*                    */
'| pad 15',                                     /*                    */
'| find' left(rem_ip,15,'_')||,                 /*                    */
'| take 1',                                     /*                    */
'| count lines',                                /*                    */
'| var auth'                                    /*                    */
                                                /*                    */
if auth = 0 then do                             /*                    */
   say my_name||hi'Unauthorized Connection Request from' rem_ip rem_port lo
   call socket 'Close',socketid                 /*                    */
   exit 999                                     /*                    */
end                                             /*                    */
```

Figure 40 (Part 2 of 16). IPGATE1I MTREXX

```
/*********************************************************************/
/* Initialize TCP/IP communication                                   */
/*********************************************************************/
init = 0
                                                /*                    */
init_data = ''                                  /*                    */
/* now run in blocking mode first ------------************************/
                                                /*                    */
do until init                                   /*                    */
   parse value Socket('Read',socketid) with rc length data
   init_data = init_data || data                /*                    */
   if length(init_data) > 13 & left(init_data,13) <> '*IPGATE*INIT*',
      then do
                                                /*                    */
      say my_name||hi'Invalid Init call from' rem_ip rem_port lo
      call socket 'Close',socketid              /*                    */
      exit 999                                  /*                    */
   end                                          /*                    */
   if words(translate(init_data,' ','*')) >= 4 then init = 1
end                                             /*                    */
                                                /*                    */
parse value socket('Write',socketid,'00'x) with rc bytes_sent .
if rc <> 0 then do                              /*                    */
   say my_name||hi'Init response write failed - rc =' rc '('bytes_sent')'lo
   call socket 'Close',socketid                 /*                    */
   exit rc                                      /*                    */
end                                             /*                    */
                                                /*                    */
parse var init_data '*IPGATE*INIT*'resource'*'user'*'ip_data
                                                /*                    */
```

Figure 40 (Part 3 of 16). IPGATE1I MTREXX

```
/* now switch to non-blocking mode -----**********************/
parse value Socket('SetSockOpt',socketid,'SOL_Socket','SO_Broadcast','On') with rc .

parse value Socket('Ioctl',socketid,'FIONBIO','On') with rc .

sel_write = ''                                  /*                    */
                                                /*                    */
call EventCreate my_name                        /*                    */
```

Figure 40 (Part 4 of 16). IPGATE1I MTREXX

```
/*********************************************************************/
/* Initialize APPC   communication                                   */
/*********************************************************************/
fill = CM_FILL_LL                               /*                    */
fill_buffer = CM_FILL_BUFFER                    /*                    */
                                                /*                    */
address 'CPICOMM' 'CMINIT cm_convid resource cm_rc'
/*********************************************************************/
                                                /*                    */
montok = EventMonitorCreate('VMCPIC' || '(' || '00000002'x || left(cm_convid,8,'_') || '*)',
                            my_name)
```

Figure 40 (Part 5 of 16). IPGATE1I MTREXX

```
/*********************************************************************/
rem_user = user                                 /*                    */
                                                /*                    */
'PIPE (end ?) < IPGATE USERMAP',                /*                    */
'| nfind ;'||,                                  /*                    */
'| gI: find' left(rem_ip,15,'_')||,
'| giU: find' left(rem_ip,15,'_') left(user,8,'_')||,
'| giuR: find' left(rem_ip,15,'_') left(user,8,'_') left(resource,8,'_')||,
'| auth: fanin 0 7 1 2 4 6 5 3',               /*                    */
'', /* iur iuR Iur IUr iUr IuR iUR IUR     uppercase = generic       */
'', /* 0   7   1   2   4   6   5   3                                 */
'| append literal x x x not_auth',             /*                    */
'| take 1',                                    /*                    */
'| specs w4 1',                                /*                    */
'| var loc_user',                              /*                    */
'? gI:',                                        /* generic ip address ?*/
'| find' left('*',15,'_')||,                   /*                    */
'| gIU: find' left('*',15,'_') left(user,8,'_')||,
'| gIuR: find' left('*',15,'_') left(user,8,'_') left(resource,8,'_')||,
'| elastic',                                   /*                    */
'| auth:',                                     /*                    */
'? gIU:',                                       /* generic ip + user ? */
'| gIUR: find' left('*',15,'_') left('*',8,'_') left(resource,8,'_')||,
'| elastic',                                   /*                    */
'| auth:',                                     /*                    */
'? gIUR:',                                      /* gen ip user reso ?  */
'| find' left('*',15,'_') left('*',8,'_') left('*',8,'_')||,
'| elastic',                                   /*                    */
'| auth:',                                     /*                    */
'? giU:',                                       /* generic user ?     */
'| find' left(rem_ip,15,'_') left('*',8,'_')||,
'| giUR: find' left(rem_ip,15,'_') left('*',8,'_') left(resource,8,'_')||,
'| elastic',                                   /*                    */
'| auth:',                                     /*                    */
'? giUR:',                                      /* gen ip +    reso ?  */
'| find' left(rem_ip,15,'_') left('*',8,'_') left('*',8,'_')||,
'| elastic',                                   /*                    */
'| auth:',                                     /*                    */
'? gIuR:',                                      /* gen reso ?         */
'| find' left('*',15,'_') left(user,8,'_') left('*',8,'_')||,
'| elastic',                                   /*                    */
'| auth:',                                     /*                    */
'? giuR:',                                      /* gen reso ?         */
'| find' left(rem_ip,15,'_') left(user,8,'_') left('*',8,'_')||,
'| elastic',                                   /*                    */
'| auth:'                                      /*                    */
                                                /*                    */
```

Figure 40 (Part 6 of 16). IPGATE1I MTREXX

```
if loc_user = 'not_auth' then do              /*                    */
                                              /*                    */
   say my_name||hi'User' rem_user 'from' rem_ip,
   'is NOT mapped for' resource lo
                                              /*                    */
   dealloc_type = CM_DEALLOCATE_ABEND         /*                    */
   address 'CPICOMM' 'CMSDT cm_convid dealloc_type rc'
   address 'CPICOMM' 'CMDEAL cm_convid rc'
                                              /*                    */
   call EventMonitorReset montok              /*                    */
   call EventMonitorDelete montok             /*                    */
                                              /*                    */
   call EventDelete my_name                   /*                    */
                                              /*                    */
   call socket 'Close',socketid               /*                    */
                                              /*                    */
   say my_name||hi'terminating.'lo            /*                    */
   exit 777                                   /*                    */
                                              /*                    */
end                                           /*                    */
                                              /*                    */
if loc_user = '=' then loc_user = rem_user    /*                    */
                                              /*                    */
if loc_user <> rem_user then do               /*                    */
   user = loc_user                            /*                    */
   say my_name||hi'User' rem_user 'from' rem_ip,
   'is mapped as' loc_user 'for' resource||lo
end                                           /*                    */
else do                                       /*                    */
   say my_name||hi'User' rem_user 'from' rem_ip,
   'has been accepted for' resource||lo
end                                           /*                    */
```

Figure 40 (Part 7 of 16). IPGATE1I MTREXX

```
/*********************************************************************/
address 'CPICOMM' 'XCSCUI cm_convid user return_code'/*set cl sec usr*/
if return_code <> 0 then do                    /*                     */
                                               /*                     */
   say my_name||hi'Unable to set Client Security User for' rem_user,
       'from' rem_ip '- terminating.'lo
                                               /*                     */
   dealloc_type = CM_DEALLOCATE_ABEND          /*                     */
   address 'CPICOMM' 'CMSDT cm_convid dealloc_type rc'
   address 'CPICOMM' 'CMDEAL cm_convid rc'
                                               /*                     */
   call EventMonitorReset montok               /*                     */
   call EventMonitorDelete montok              /*                     */
                                               /*                     */
   call EventDelete my_name                    /*                     */
                                               /*                     */
   call socket 'Close',socketid                /*                     */
                                               /*                     */
   exit 666                                    /*                     */
                                               /*                     */
end                                            /*                     */
                                               /*                     */
conv_type = CM_BASIC_CONVERSATION              /*                     */
address 'CPICOMM' 'CMSCT cm_convid conv_type rc'
                                               /*                     */
address 'CPICOMM' 'CMALLC cm_convid rc'        /*                     */
                                               /*                     */
send_type = CM_SEND_AND_PREP_TO_RECEIVE        /*                     */
address 'CPICOMM' 'CMSST cm_convid send_type rc'
/*********************************************************************/
ip_read_done = 0                               /*                     */
```

Figure 40 (Part 8 of 16). IPGATE1I MTREXX

```
/*********************************************************************/
parse value socket('Select','Read' socketid 'Write' sel_write ,
                   'Exception' socketid,'SIGNAL' my_name,) with rc
                                          /*                       */
finis = 0                                 /*                       */
                                          /*                       */
text = ''                                 /*                       */
tcp_text = ''                             /*                       */
                                          /*                       */
$REM_INFO$ = left(rem_user,8) rem_ip rem_port resource
$LOC_INFO$ = left(loc_user,8) socketid date() time()
$IP_DATA_OUT$ = 0                         /*                       */
$IP_DATA_IN$ = 0                          /*                       */
$LAST_ACTIV$ = date() time()              /*                       */
                                          /*                       */
do until finis                            /*                       */
   call EventWait montok                  /* wait for event        */
                                          /*                       */
   if retcode <> 0 then finis = 1         /*                       */
                                          /*                       */
$LAST_ACTIV$ = date() time()              /*                       */
                                          /*                       */
```

Figure 40 (Part 9 of 16). IPGATE1I MTREXX

```
/*-----------------------------------------*************************/
   if event_flag.1 >= 0 then do              /* VMCPIC               */
                                             /*                      */
      call EventRetrieve montok 1 32767      /*                      */
                                             /*                      */
      if left(retdata,4) = '00000002'x then do /* Information input */
         parse var retdata . 5 conv_id 13 length +4 .
         total_sent = c2d(length)            /*                      */
         total_received = 0                  /*                      */
/*-----------------------------------------*************************/
                                             /*                      */
         if length = 0 then finis = 1        /*                      */
/*-----------------------------------------*************************/
         else do until total_received >= total_sent
            receive_type = CM_RECEIVE_AND_WAIT/*                     */
                                             /*                      */
         address 'CPICOMM' 'CMECS cm_convid conversation_state return_code'
                                             /*                      */
if cm_conversation_state.conversation_state <> 'CM_RECEIVE_STATE' then,
   say my_name 'Rcv_state' cm_conversation_state.conversation_state
                                             /*                      */
            address 'CPICOMM' 'CMSRT cm_convid receive_type return_code'
            requested_length = c2d(length)
            if requested_length > 32767 then requested_length = 32767
                                             /*                      */
if return_code <> 0 then say my_name 'CMSRT' return_code
                                             /*                      */
            r_buffer = ''                    /*                      */
                                             /*                      */
            address 'CPICOMM' 'CMRCV cm_convid r_buffer requested_length',
                                'data_received received_length status_received',
                                'request_to_send_received return_code'
if return_code <> 0 then say my_name 'CMRCV' cm_return_code.return_code
            if cm_return_code.return_code = 'CM_RESOURCE_FAILURE_NO_RETRY' ,
             | cm_return_code.return_code = 'CM_DEALLOCATED_ABEND' then do
               dealloc_type = CM_DEALLOCATE_ABEND
               address 'CPICOMM' 'CMSDT cm_convid dealloc_type rc'
               address 'CPICOMM' 'CMDEAL cm_convid rc'
                                             /*                      */
               finis = 1                     /*                      */
               leave                         /*                      */
            end                              /*                      */
            total_received = total_received + received_length
                                             /*                      */
            send_type = CM_BUFFER_DATA       /*                      */
                                             /*                      */
            if total_received >= total_sent & ,
               cm_status_received.status_received <> 'CM_NO_STATUS_RECEIVED' then ,
               send_type = CM_SEND_AND_PREP_TO_RECEIVE
                                             /*                      */
            text = text'*IPGATE*CMSST*'send_type'*'
                                             /*                      */
```

Figure 40 (Part 10 of 16). IPGATE1I MTREXX

```
            s_buffer = left(r_buffer,received_length)
            send_length = c2d(left(r_buffer,2))
                                          /*                      */
            text = text || '*IPGATE*CMSEND*'||length(s_buffer)||'*'||,
                   s_buffer||'*'
                                          /*                      */
            if sel_write = '' then do     /*                      */
               wr_text = left(text,min(50000,length(text)))
               parse value socket('Write',socketid,wr_text) with rc bytes_sent .
                                            /*                    */
               if rc <> 0 & rc <> 35 then do /*                    */
                  finis = 1                 /*                    */
                  say my_name||hi'write failed with rc =' rc '('bytes_sent')'lo
               end                          /*                    */
               if rc = 35 then do
                  if sel_write = '' then do /*                    */
                     sel_write = socketid /* EWOULDBLOCK      */
                     parse value socket('Select','Read',
                     'Write' sel_write ,
                     'Exception','SIGNAL' my_name) with sel_rc .
                  end                       /*                    */
               end                          /*                    */
               if rc = 0 then do
                  if bytes_sent = '' then do /*                    */
                     say my_name||hi'Socket Write failed'lo
                     call ThreadDelete '-1'  /*                    */
                     exit 1111               /*                    */
                  end                        /*                    */
                  $IP_DATA_OUT$ = $IP_DATA_OUT$ + bytes_sent
                  if bytes_sent = length(text) then do
                     text = ''               /*                    */
                     sel_write = ''          /*                    */
                  end                        /*                    */
                  else do                    /*                    */
                     text = substr(text,bytes_sent+1)
                     if sel_write = '' then do  /*                    */
                        sel_write = socketid /* EWOULDBLOCK      */
                        parse value socket('Select','Read',
                        'Write' sel_write ,
                        'Exception','SIGNAL' my_name) with rc .
                     end                     /*                    */
                  end                        /*                    */
               end                           /*                    */
            end                              /*                    */
         end                                 /*                    */
/*-----------------------------------------*************************/
      end                                    /*                    */
      else do                                /*                    */
         say my_name||hi'Unexpected APPC input ('retdata '-' c2x(retdata)')',
                   'or termination request'lo
         finis = 1                           /*                    */
      end                                    /*                    */
   end                                       /*                    */
```

Figure 40 (Part 11 of 16). IPGATE1I MTREXX

```
/*-----------------------------------------*************************/
   if event_flag.2 >= 0 then do              /* TCP/IP               */
                                             /*                      */
      ip_read_done = 0                       /*                      */
                                             /*                      */
      read_rc = 0                            /*                      */
                                             /*                      */
      call EventRetrieve montok 2 32767      /*                      */

      parse var retdata tcp_events ' READ ' tcp_read ' WRITE ' tcp_write ,
                        ' EXCEPTION ' tcp_except
                                             /*                      */
if (tcp_read <> socketid) & tcp_read <> '' then ,
   say my_name||hi'Read for' tcp_read 'signaled - should be' socketid||lo
if (tcp_write <> socketid) & tcp_write <> '' then ,
   say my_name||hi'Write for' tcp_write 'signaled - should be' socketid||lo
if (tcp_except <> socketid) & tcp_except <> '' then ,
   say my_name||hi'Exception for' tcp_except 'signaled - should be' socketid||lo
```

Figure 40 (Part 12 of 16). IPGATE1I MTREXX

```
/*- receiving tcp answer ---------------------*************************/
      if tcp_read <> '' then do
         parse value socket('Read',socketid,100000) with rc bytes data
         read_rc = rc                        /*                      */
         if rc <> 35 & (rc <> 0 | bytes = 0) then do
            say my_name||hi'Thread terminating ... Read rc =' rc '('bytes data')'lo
            finis = 1
         end
```

Figure 40 (Part 13 of 16). IPGATE1I MTREXX

```
         else if rc <> 35 then do until ip_read_done
            $IP_DATA_IN$ = $IP_DATA_IN$ + bytes
            tcp_text = tcp_text || data       /*                      */
            data = ''                         /*                      */
            select                            /*                      */
/*- set send type received *IPGATE*CMSST*n* --***********************/
               when left(tcp_text,14) = '*IPGATE*CMSST*',
                  & length(tcp_text) >= 16 then do
                                              /*                      */
                  parse var tcp_text '*IPGATE*CMSST*'send_type'*'tcp_text
                  address 'CPICOMM' 'CMSST cm_convid send_type rc'
               end                            /*                      */
/*- send data received *IPGATE*CMSEND*length*data* ******************/
               when left(tcp_text,15) = '*IPGATE*CMSEND*',
                  & length(tcp_text) >= 19 then do
                                              /*                      */
                  parse var tcp_text '*IPGATE*CMSEND*'tcp_temp
                  if index(tcp_temp,'*') > 0 then do
                                              /*                      */
                     parse var tcp_text '*IPGATE*CMSEND*'length'*'tcp_temp
if length = 0 then say my_name||hi'ip-cmsend length = 0'lo
                     if length(tcp_temp) >= length + 1 then do /* data complete ? */
                                              /*                      */
                        cm_data = left(tcp_temp,length)
                        if length(tcp_temp) = length + 1 then,
                           tcp_text = ''      /*                      */
                        else tcp_text = substr(tcp_temp,length+2)
                                              /*                      */
                        address 'CPICOMM' 'CMECS cm_convid',
                                'conversation_state return_code'
                                              /*                      */
if cm_conversation_state.conversation_state <> 'CM_SEND_STATE' &,
   cm_conversation_state.conversation_state <> 'CM_INITIALIZE_STATE' &,
   cm_conversation_state.conversation_state <> 'CM_SEND_PENDING_STATE' then,
   say my_name 'Send_state' cm_conversation_state.conversation_state
                                              /*                      */
if cm_conversation_state.conversation_state = 'CM_INITIALIZE_STATE' then,
               finis = 1                      /*                      */
                                              /*                      */
                        address 'CPICOMM' 'CMSEND cm_convid cm_data',
                                          'length req_to_send rc'
                                              /*                      */
if rc <> 0 then say my_name 'CMSEND' cm_return_dode.rc
                     end                      /*                      */
                     else ip_read_done = 1    /*                      */
                  end                         /*                      */
                  else ip_read_done = 1       /*                      */
               end                            /*                      */
/*- else wait for complete data --------------***********************/
               otherwise ip_read_done = 1     /*                      */
            end                               /*                      */
         end
      end
```

Figure 40 (Part 14 of 16). IPGATE1I MTREXX

```
/*-----------------------------------------*************************/
/* if write is signaled, the remaining data to send must be in       */
/* variable text                                                     */
/*-----------------------------------------*************************/
      if tcp_write <> '' then do             /*                      */
         if text = '' then do                /* nothing to write ?  */
            sel_write = ''                   /*                      */
         end                                 /*                      */
         else do                             /*                      */
            wr_text = left(text,min(50000,length(text)))
            parse value socket('Write',socketid,wr_text) with rc bytes_sent .
/*-----------------------------------------*************************/
            if rc <> 0 & rc <> 35 then do    /*                      */
               finis = 1                     /*                      */
               say my_name 'write failed with rc =' rc '('bytes_sent')'lo
            end                              /*                      */
            if rc = 35 then do
               if sel_write = '' then do     /*                      */
                  sel_write = socketid /* EWOULDBLOCK      */
               end                           /*                      */
            end                              /*                      */
            if rc = 0 then do
               $IP_DATA_OUT$ = $IP_DATA_OUT$ + bytes_sent
               if bytes_sent = length(text) then do
                  text = ''                  /*                      */
                  sel_write = ''             /*                      */
               end                           /*                      */
               else do                       /*                      */
                  text = substr(text,bytes_sent+1)
                  if sel_write = '' then do      /*                    */
                     sel_write = socketid /* EWOULDBLOCK      */
                  end                            /*                    */
               end                           /*                      */
            end                              /*                      */
/*-----------------------------------------*************************/
         end                                 /*                      */
      end
```

Figure 40 (Part 15 of 16). IPGATE1I MTREXX

```
/*-----------------------------------------*************************/
      if tcp_except <> '' then do
         parse value socket('Recv',socketid,,'OUT_OF_BAND') with rc bytes data
         say my_name 'exception read' bytes 'bytes.'
         say my_name 'exception data:' c2x(data)
      end
      if finis = 0  then ,                  /*                     */
      parse value socket('Select','Read' socketid 'Write' sel_write ,
                         'Exception' socketid,'SIGNAL' my_name,) with rc
   end                                      /*                     */
end                                         /*                     */
/*-----------------------------------------*************************/
dealloc_type = CM_DEALLOCATE_ABEND          /*                     */
address 'CPICOMM' 'CMSDT cm_convid dealloc_type rc'
address 'CPICOMM' 'CMDEAL cm_convid rc'
                                            /*                     */
call EventMonitorReset montok               /*                     */
call EventMonitorDelete montok              /*                     */
                                            /*                     */
call socket 'Close',socketid                /*                     */
                                            /*                     */
call EventDelete my_name                    /*                     */
                                            /*                     */
say my_name 'ended.'                        /*                     */
                                            /*                     */
exit                                        /*                     */
```

Figure 40 (Part 16 of 16). IPGATE1I MTREXX

4.6.7 IPGATE1W MTREXX - Monitors APPC Requests for a Resource

```
/*********************************************************************/
/*                                                                   */
/* Allocation Request                                                */
/*                                                                   */
/* +-------------+-----------------+                                 */
/* | X'00000001' |   resource_id   |                                 */
/* +-------------+-----------------+                                 */
/*    4 bytes       8 bytes                                          */
/*                                                                   */
/*                                                                   */
/* Information Input                                                 */
/*                                                                   */
/* +-------------+-----------------+------------------+              */
/* | X'00000002' | conversation_id | event_info_length |              */
/* +-------------+-----------------+------------------+              */
/*    4 bytes       8 bytes            4 bytes                       */
/*                                                                   */
/*                                                                   */
/* Resource revoked notification                                     */
/*                                                                   */
/* +-------------+-----------------+                                 */
/* | X'00000003' |   resource_id   |                                 */
/* +-------------+-----------------+                                 */
/*    4 bytes       8 bytes                                          */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
address 'COMMAND'                              /*                   */
                                               /*                   */
parse value '1D6O4O1DE8'x with lo hi .         /*                   */
                                               /*                   */
call MT_Init                                   /*                   */
                                               /*                   */
parse arg args                                 /*                   */
parse var args my_resource target_resource scope .
                                               /*                   */
call ThreadGetId                               /*                   */
                                               /*                   */
my_name = 'IPGATEW' || right(thread_ID,1O,O)
                                               /*                   */
say my_name 'started ('args')'
                                               /*                   */
```

Figure 41 (Part 1 of 4). IPGATE1W MTREXX

```
call IdentifyResourceManager my_resource value('XC_' ||  scope)
                                          /*                      */
if return_code <> 0 then do               /*                      */
   say 'IdentifyResourceManager for resource' my_resource,
       'failed with rc =' return_code     /*                      */
   exit return_code                       /*                      */
end                                       /*                      */
                                          /*                      */
say my_name 'IdentifyResourceManager for' my_resource 'successful'
                                          /*                      */
montok = EventMonitorCreate('VMCPIC' || '(%%%%' || left(my_resource,8,'_') || '*)')
                                          /*                      */
finis = 0                                 /*                      */
                                          /*                      */
do until finis                            /*                      */
   call EventWait montok                  /* wait for event       */
                                          /*                      */
   if retcode <> 0 then finis = 1         /*                      */
                                          /*                      */
```

Figure 41 (Part 2 of 4). IPGATE1W MTREXX

```
   if event_flag.1 >= 0 then do           /* VMCPIC               */
                                          /*                      */
      call EventRetrieve montok 1 32767   /*                      */
                                          /*                      */
      select                              /*                      */
/*----------------------------------------**********************/
         when left(retdata,4) = '00000001'x then do  /* Allocation request */
                                          /*                      */
            parse var retdata . 5 resource_name +8 .
            parse value ThreadCreate('IPGATE1Y' resource_name) with rc .
         end                              /*                      */
/*----------------------------------------**********************/
         when left(retdata,4) = '00000002'x then do /* Information input */
                                          /*                      */
            say my_name||hi'Information input ??????????'lo
         end                              /*                      */
/*----------------------------------------**********************/
         when left(retdata,4) = '00000003'x then do /* Resource revoked notification */
            say my_name||hi'Resource' substr(retdata,5) 'revoked.'lo
            finis = 1                     /*                      */
         end                              /*                      */
/*----------------------------------------**********************/
         otherwise do                     /*                      */
            say my_name||hi'Unexpected event or terminate request'||lo
            finis = 1                     /*                      */
         end                              /*                      */
      end                                 /*                      */
   end                                    /*                      */
```

Figure 41 (Part 3 of 4). IPGATE1W MTREXX

```
   if event_flag.2 >= 0 then do              /* VMCPIC             */
      say my_name||hi' Information Available received' lo
   end                                       /*                    */
end                                          /*                    */
                                             /*                    */
call EventDelete my_name                     /*                    */
                                             /*                    */
call EventMonitorReset montok                /*                    */
call EventMonitorDelete montok               /*                    */
                                             /*                    */
call TerminateResourceManager my_resource    /*                    */
say my_name 'ended.'                         /*                    */
                                             /*                    */
exit                                         /*                    */
```

Figure 41 (Part 4 of 4). IPGATE1W MTREXX

4.6.8 IPGATE Subroutines

```
/*********************************************************************/
MT_Init:                                     /*                    */
                                             /*                    */
call apiload 'CMREXX'                        /*                    */
call apiload 'VMREXMT'                       /*                    */
call apiload 'VMREXRET'                      /*                    */
                                             /*                    */
return                                       /*                    */
```

Figure 42. IPGATE Subroutine MT_Init

```
/*********************************************************************/
EventCreate:                                 /*                    */
                                             /*                    */
arg event_name                               /*                    */
                                             /*                    */
event_name_length = length(event_name)       /*                    */
                                             /*                    */
event_flag.0 = 3                             /*                    */
event_flag.1 = vm_evn_session_scope          /*                    */
event_flag.2 = vm_evn_broadcast_signals      /*                    */
event_flag.3 = vm_evn_async_signals          /*                    */
event_flag_size = 3                          /*                    */
                                             /*                    */
loose_signal_limit = 0                       /*                    */
                                             /*                    */
signal_timeout_period = 0                    /*                    */
                                             /*                    */
call csl 'EventCreate retcode reascode event_name event_name_length',
         'event_flag event_flag_size loose_signal_limit',
         'signal_timeout_period'
                                             /*                    */
return                                       /*                    */
```

Figure 43. IPGATE Subroutine EventCreate

```
/*********************************************************************/
EventDelete:                                /*                     */
                                            /*                     */
arg event_name                              /*                     */
                                            /*                     */
event_name_length = length(event_name)      /*                     */
                                            /*                     */
call csl 'EventDelete retcode reascode event_name event_name_length'
                                            /*                     */
return                                      /*                     */
```

Figure 44. IPGATE Subroutine EventDelete

```
/*********************************************************************/
EventRetrieve:                              /*                     */
                                            /*                     */
parse arg monitor_token index retlen        /*                     */
                                            /*                     */
if retlen = '' then retlen = 200            /*                     */
                                            /*                     */
call csl,                                   /*                     */
 'EventRetrieve ',     /* routine name  */  /*                     */
 'retcode',            /* return code   */  /*                     */
 'reascode',           /* reason code   */  /*                     */
 'monitor_token',      /* monitor token */  /*                     */
 'index',              /* event index   */  /*                     */
 'retdata ',           /* returned data */  /*                     */
 'event_flag.'index,   /* max to return */  /*                     */
 'retlen '             /* length returned */ /*                   */
                                            /*                     */
return                                      /*                     */
```

Figure 45. IPGATE Subroutine EventRetrieve

```
/*********************************************************************/
EventWait:                                  /*                     */
                                            /*                     */
parse arg monitor_token                     /*                     */
                                            /*                     */
call csl 'EventWait retcode reascode monitor_token number_of_events',
         'event_flag'                       /*                     */
                                            /*                     */
return number_of_events                     /*                     */
```

Figure 46. IPGATE Subroutine EventWait

```
/*********************************************************************/
/* EventMonitorCreate('VMCON1ECB' 'VMTIMER(%%%%E' || userword)       */
/*********************************************************************/
EventMonitorCreate:                          /*                     */
                                             /*                     */
parse arg events                             /*                     */
                                             /*                     */
monitor_flag.1 = vm_evn_no_auto_delete       /*                     */
monitor_flag.2 = vm_evn_async_monitor        /*                     */
monitor_flag.3 = vm_evn_bind_loose_signals   /*                     */
                                             /*                     */
monitor_flag_size = 3                        /*                     */
                                             /*                     */
number_of_events = words(events)             /*                     */
                                             /*                     */
do i = 1 to words(events)                    /*                     */
                                             /*                     */
   event_name_address = ''                   /*                     */
   event_key_address = '*'                   /*                     */
                                             /*                     */
   parse value word(events,i) with event_name_address '(' event_key_address ')'
                                             /*                     */
                                             /*                     */
                                             /*                     */
   event_name_address.i = event_name_address /*                     */
                                             /*                     */
   event_name_length.i = length(event_name_address.i)
                                             /*                     */
   event_key_address.i = event_key_address   /*                     */
                                             /*                     */
   event_key_length.i = length(event_key_address.i)
                                             /*                     */
   bound_signal_limit.i = -1                 /*                     */
                                             /*                     */
end                                          /*                     */
                                             /*                     */
event_count = 1                              /*                     */
                                             /*                     */
call csl 'EventMonitorCreate retcode reascode monitor_token',
         'monitor_flag monitor_flag_size number_of_events',
         'event_name_address event_name_length event_key_address',
         'event_key_length bound_signal_limit event_count'
                                             /*                     */
return monitor_token                         /*                     */
```

Figure 47. IPGATE Subroutine EventMonitorCreate

```
/*********************************************************************/
EventMonitorDelete:                          /*                     */
                                             /*                     */
arg mon_token                                /*                     */
                                             /*                     */
call csl 'EventMonitorDelete retcode reascode mon_token'
                                             /*                     */
return                                       /*                     */
```

Figure 48. IPGATE Subroutine EventMonitorDelete

```
/*********************************************************************/
EventMonitorReset:                           /*                     */
                                             /*                     */
arg mon_token                                /*                     */
                                             /*                     */
call csl 'EventMonitorReset retcode reascode mon_token'
                                             /*                     */
return                                       /*                     */
```

Figure 49. IPGATE Subroutine EventMonitorReset

```
/*********************************************************************/
IdentifyResourceManager:                     /*                     */
                                             /*                     */
arg resource_ID resource_manager_type service_mode security_level_flag
if resource_ID = '' then return
if resource_manager_type = '' then resource_manager_type = XC_SYSTEM
if service_mode = '' then service_mode = XC_MULTIPLE
if security_level_flag = '' then security_level_flag = XC_REJECT_SECURITY_NONE
                                             /*                     */
address 'CPICOMM' 'XCIDRM resource_ID resource_manager_type',
                  'service_mode security_level_flag return_code'
                                             /*                     */
return                                       /*                     */
```

Figure 50. IPGATE Subroutine IdentifyResourceManager

```
/*********************************************************************/
TerminateResourceManager:                    /*                     */
                                             /*                     */
arg resource_ID                              /*                     */
                                             /*                     */
address 'CPICOMM' 'XCTRRM resource_ID return_code'
                                             /*                     */
return                                       /*                     */
```

Figure 51. IPGATE Subroutine TerminateResourceManager

```
/*********************************************************************/
/* ThreadDelay(1000)                                                 */
/*********************************************************************/
ThreadDelay:                                /*                    */
                                            /*                    */
parse arg msecs .                           /*                    */
                                            /*                    */
call csl 'ThreadDelay retcode reascode' msecs /*                   */
                                            /*                    */
return                                      /*                    */
```

Figure 52. IPGATE Subroutine ThreadDelay

```
/*********************************************************************/
ThreadDelete:                               /*                    */
                                            /*                    */
arg thread .                                /*                    */
                                            /*                    */
call csl 'ThreadDelete retcode reascode thread'
                                            /*                    */
return                                      /*                    */
```

Figure 53. IPGATE Subroutine ThreadDelete

```
/*********************************************************************/
ThreadGetId:                                /*                    */
                                            /*                    */
call csl 'ThreadGetId retcode reascode thread_ID process_ID'
                                            /*                    */
return                                      /*                    */
```

Figure 54. IPGATE Subroutine ThreadGetID

```
/*********************************************************************/
ThreadSetPriority:                          /*                    */
                                            /*                    */
arg thread priority .                       /*                    */
                                            /*                    */
if priority = '' then priority = 32767      /*                    */
                                            /*                    */
flags = vm_pro_absolute_priority            /*                    */
                                            /*                    */
call csl 'ThreadSetPriority retcode reascode thread priority flags'
                                            /*                    */
return                                      /*                    */
```

Figure 55. IPGATE Subroutine ThreadSetPriority

```
/*********************************************************************/
ThreadYield:                                /*                     */
                                            /*                     */
arg thread2yield .                          /*                     */
if thread2yield = '' then thread2yield = 0  /*                     */
                                            /*                     */
call csl 'ThreadYield retcode reascode thread2yield'
                                            /*                     */
return                                      /*                     */
```

Figure 56. IPGATE Subroutine ThreadYield

```
/*********************************************************************/
/* TimerStartInt_Single(5 TIMER)                                     */
/*********************************************************************/
TimerStartInt_Single:                       /*                     */
                                            /*                     */
arg seconds userword                        /*                     */
                                            /*                     */
timertype = vm_tmr_timertype_real           /*                     */
cycle = vm_tmr_cycle_single                 /*                     */
intervalunits = vm_tmr_intunit_milli        /*                     */
interval = seconds * 1000                   /* number of seconds   */
                                            /*                     */
if symbol('userword') <> 'VAR' | userword = '' then ,
   userword = copies('00'x,8)               /*                     */
else userword = left(userword,8)            /*                     */
                                            /*                     */
call csl 'TimerStartInt retcode reascode timer_token',
         'timertype cycle intervalunits interval userword'
                                            /*                     */
return                                      /*                     */
```

Figure 57. IPGATE Subroutine TimerStartInt_Single

Appendix A. Supplementary Information on System Defined Events

The names of system defined events are well documented in CMS Application Multitasking. However, it is sometimes difficult to find documentation defining the characteristics used when these events were defined and what, if any, keys will be used when the events are signaled. We have consolidated that information here for all system events that were documented in CMS release 14.

A.1 System Event Characteristics

Table 1. System Event Characteristics

Name	Scope	Signal delivery	Signaler treat-ment	Loose signal limit	Signal timeout period	Notes
VMACCOUNT	Session	Broadcast	Async	-1	0	Must enable with AccountControl
VMCONINPUT	Session	Broadcast	Async	0	0	
VMCON1ECB	Session	Broadcast	Async	0	0	
VMCPIC	Session	Broadcast	Async	0	0	
VMERROR	Process	LIFO	Sync Process	0	0	
VMERRORCHILD	Process	LIFO	Sync Process	0	0	
VMIPC	Process	Broadcast	Async	0	0	Must enable with QueueSignalEvents
VMPOSGNL	Process	Broadcast	Async	0	0	Only created for POSIX processes
VMPROCESSEND	Session	Broadcast	Sync Thread	0	0	
VMSFSASYNC	Session	Broadcast	Sync Thread	0	0	
VMSOCKET	Session	Broadcast	Async	1	0	Only defined while RXSOCKET is active
VMTIMECHANGE	Session	Broadcast	Async	0	0	
VMTIMER	Session	Broadcast	Async	-1	0	
VMTRACE	Session	Broadcast	Async	50	0	See also the CMS command TRACECTL

A.2 System Event Signal Data/Key Information

When an event is signaled, the signaler may also include data as part of the signal. Additionally that data may also contain a key to help with selectively handling signals. Information relating to system event's signal data and their keys (if any) has been consolidated here. Unless otherwise specified documentation references are to *CMS Application Multitasking*, SC24-5766.

Table 2. System Event Key Information

Name	Signal data?	Key with signal?	Signal data format/key documented in:
VMACCOUNT	Yes	Yes	Accounting services chapter
VMCONINPUT	No	No	
VMCON1ECB	No	No	
VMCPIC	Yes	Yes	*SAA CPI-C Reference*, SC26-4399-08; *VM/ESA CPI-C User's Guide*, SC24-5595-01
VMERROR	Yes	Yes	Abend services chapter
VMERRORCHILD	Yes	Yes	Abend services chapter
VMIPC	Yes	Yes	Usage notes to QueueSignalEvents
VMPOSGNL	Yes	Yes	OpenEdition services appendix
VMPROCESSEND	Yes	Yes	Process management chapter
VMSFSASYNC	Yes	Yes	*CMS Application Development Guide*, SC24-5761
VMSOCKET	Yes	No	See below
VMTIMECHANGE	Yes	Yes	Timer services chapter
VMTIMER	Yes	Yes	Timer services chapter
VMTRACE	Yes	Yes	Trace services chapter

A.3 VMCONINPUT versus VMCON1ECB

Two of the system events pertain to console activity:

VMCONINPUT unsolicited attention received at the console

VMCON1ECB input available at the console

One question you no doubt have is: *What is the difference between VMCONINPUT and VMCON1ECB?*

For many applications VMCONINPUT and VMCON1ECB may be used interchangeably. Every time an unsolicited attention interrupt is received on the console, VMCONINPUT is signaled. However, the data represented by the signal may be consumed by CMS before the application has a chance to read it (for example, an immediate command was entered).

VMCON1ECB is signaled less frequently than VMCONINPUT. VMCON1ECB is only signaled when data is available for the application to read (it is not signaled for immediate commands, for example). When VMCON1ECB is signaled the

application is guaranteed that if it issues a read the virtual machine will not end up in VM READ.

A.4 VMSOCKET Signal Data

VMSOCKET is signaled by REXX Sockets when a select request completes. What normally would have been returned from a Select subfunction, namely a count of completed events followed by a list of the events, is provided as the signal data. The original documentation in the CMS 13 Application Development Reference describing this function and its syntax was incorrect. Correct documentation can be found in the CMS 14 level of the *REXX/VM Reference*, SC24-5770-02.

Most system events provide signal data. VMSOCKET is unique among system defined events in that it provides signal data but no key. Broadly speaking, with the system events, the key is used to provide a degree of uniqueness between events. Rather than a key REXX Sockets allows a unique event name to be provided for each outstanding Select request. Chapter 4, “Sample Application IPGATE” on page 51 makes use of this capability and serves as an example.

One final note, if when the socket Terminate function is called, there remain outstanding Selects, they are signaled with a completed event count of zero.

Appendix B. Supplementary Information for REXX Programmers

B.1 ThreadDelete Caution

The use of ThreadDelete to terminate REXX threads is strongly discouraged. When a REXX thread is deleted, the REXX interpreter (or compiler runtime) is not notified that the thread has been deleted. Thus there is no opportunity for cleanup to occur, and any storage used in support of that thread will not be released (storage is obtained not only for variables but may also be required to contain the REXX program itself). Repeated use of ThreadDelete will lead to storage exhaustion. The storage will be full of obsolete REXX programs and variables (a storage cancer).

B.2 APILOAD Caution

The function routine names and associated constant names and values are defined in *binding files*. Use of the associated names, as defined in the binding files, is encouraged. In most languages, binding files are included by the compiler at compile time. With REXX, access to the binding files occurs at execution time by invoking the APILOAD function. On CMS level 11 and earlier the use of APILOAD can prove to be very costly. For some applications the resource consumption of APILOAD may exceed that of the application itself! In some cases it may be necessary to avoid using APILOAD to obtain the required values. If this must be done simply imbed the contents of the needed binding files within your application. This is easy to do since they are syntactically valid REXX.

For CMS level 12 APILOAD was redeveloped and, on CMS 12 and subsequent releases, the resource consumption is very much improved. On these current CMS releases APILOAD's performance should be adequate for most applications. There may still be situations in which it is appropriate to perform the minimum required setup manually, such as Figure 1 on page 5 or Figure 17 on page 32, but they arise much less frequently on CMS 12 than before. For all but the most performance critical applications, and especially for longer running or more complex threads, it makes more sense to APILOAD the binding files. APILOADing only the required binding files will still help to keep resource consumption to a minimum.

B.3 Constants on CSL Calls

Simple constants that are defined only for input to a function, rather than defined as being for input and output or output only, may be placed as quoted literals in the CSL call. You do not need to place them into a REXX variable. The REXX CSL interface routine recognizes this special case. For example:

```
call csl 'ThreadDelay retcode reascode "500"'
```

B.4 RXSOCKET must be at Level 3.02

In order to have full CMS MT support RXSOCKET must be at level 3.02. APAR VM61811 addresses a number of CMS MT related problems. It also changes RXSOCKET's code level to 3.02 so it is possible for an MT application to verify it is using an RXSOCKET that contains suitable support. Here is an example of using the Version subfunction to verify the software level:

```
...
parse value socket('Version') with . . level .
if level \= '3.02' then
...
```

B.5 Multithreaded Debugging Strategies

Debugging a multithreaded REXX application can be more challenging than debugging a typical single threaded application. The usefulness of REXX's excellent interactive debug begins to break down or can be confusing in a multithreaded environment. Here are a few debugging techniques we found helpful:

- Debug with as few threads as possible.
  If you know there is a problem, try and reproduce it with as few threads as possible. Running multiple threads, each with tracing or debug output merely adds confusion.
- Add thread identification to debug output
  Adding says to REXX code is a tried and true debugging technique that breaks down in a multithreaded environment. Additional information should be added to debug says to assist you in determining which thread has issued the say. At the very least the thread's name should be included on all debug output. This can easily be obtained with a `parse source` statement. If the application runs multiple copies of the same thread it may be useful to include the thread ID in addition to the thread name. This can easily be obtained with a `ThreadGetID` function. The IPGATE application in Chapter 4, "Sample Application IPGATE" on page 51 includes the thread name and ID on all messages it produces. It also uses the combination of thread name and ID to provide event names that are unique within the process.
- Thread switches can occur on function calls.
  Remember that on some CMS MT CSL calls (or RXSOCKET calls or CPIC calls) a thread switch can occur. This can be particularly surprising when single stepping through a program. The thread you were stepping through when the program initiated a function call may not be the same one you are stepping through on the next debug read from REXX. You might be tipped off that a thread switch has occurred if the line numbers on the REXX debug output are not in sequence, or the number of results returned by the function is not as expected. However, it can become very confusing if you are debugging multiple copies of the same thread.
- RXSOCKET DEBUG NOTRACE
  RXSOCKET V3 has always had an undocumented DEBUG option. It provides extensive debug information on RXSOCKET internal processing that is only of interest when debugging RXSOCKET itself. In addition to the internal RXSOCKET tracing, DEBUG also displays the REXX source string, the current REXX clause for each RXSOCKET function call and whether the function call caused the thread to block.

This REXX related debug information is useful for the REXX programmer but it was overwhelmed by the internal RXSOCKET tracing. RXSOCKET, at the 3.02 level, enhances DEBUG with the NOTRACE option. DEBUG NOTRACE provides only the REXX relevant information without the RXSOCKET internal trace. DEBUG NOTRACE is not passed to RXSOCKET via a function call, but rather, via CMS command line parameters. You activate by entering:

```
RXSOCKET DEBUG NOTRACE
```

from the CMS command line before invoking your application.

Appendix C. Supplementary Information for Assembler Programmers

C.1 CSL Call Choices

All CMS MT functions are invoked via CSL calls. Assembler programmers have three distinct methods of calling CSL routines available to them:

- Call via DMSCSL
- Fastpath call with CSLFPI
- Direct call

Of these three only direct call should be used to invoke CMS MT routines. It is the only interface of the three that is thread safe (that is to say, can tolerate a thread switch while a call is active). Although the other calling techniques might appear to work correctly, they will result, especially under load, in ABENDs or unusual and difficult to diagnosis failures such as storage overlays or threads running with the wrong context. Additionally of the three, direct call is the best performer so there is really no reason to consider either of the other two.

C.2 Binding Files

Be aware that including any binding file will cause an EXTRN to be generated for each routine it defines. Including VMASMMT then, will EXTRN all CMS MT routines. This causes the loader to include them at linkage edit time regardless of whether they are actually used or not. This can increase the size of the module unnecessarily. One option is to specify WEAK=YES when including the binding file. This option will cause WXTRNs rather than EXTRNs to be generated. Weak externals do not resolve automatically at linkage edit time, so each required routine must be manually included.

There is no automatic way to include only the routines required. For example, WXTRNs can not be automatically "upgraded" to EXTRNs by referencing them with a V-type address constant (not that one wants to use a V-type address constant anyway since the symbol specified is not subject to expansion via EQUs meaning descriptive function names could not be used).

The only method available to force automatic routine inclusion is to forgo using the binding files and manually define the required routines. WEAK=NONE would have been a nice option (allowing the binding file to be included but requiring the programmer to manually EXTRN only those routines required).

Appendix D. Special Notices

This publication is intended to help VM/ESA technical professionals and programmers to write and deploy CMS applications using the facilities of CMS Application Multitasking. The information in this publication is not intended as the specification of any programming interfaces that are provided by VM/ESA Version 2 Release 3. See the PUBLICATIONS section of the IBM Programming Announcement for VM/ESA Version 2 Release 3 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM
RACF
Virtual Machine/Enterprise Systems Architecture
VTAM
OpenEdition
SAA
VM/ESA

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

Intel is a trademark of Intel Corporation.

Java and HotJava are trademarks of Sun Microsystems, Incorporated.

Microsoft, Windows, Windows NT, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

Pentium, MMX, ProShare, LANDesk, and ActionMedia are trademarks or registered trademarks of Intel Corporation in the U.S. and other countries.

POSIX is a trademark of Institute of Electrical and Electronic Engineers.

Sun Microsystems is a trademark of Sun Microsystems, Incorporated.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

X/Open is a trademark of X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Appendix E. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

E.1 International Technical Support Organization Publications

For information on ordering ITSO publications see “How to Get ITSO Redbooks” on page 109.

E.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
Lotus Redbooks Collection	SBOF-6899	SK2T-8039
Tivoli Redbooks Collection	SBOF-6898	SK2T-8044
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RS/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RS/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
RS/6000 Redbooks Collection (PDF Format)	SBOF-8700	SK2T-8043
Application Development Redbooks Collection	SBOF-7290	SK2T-8037

E.3 Other Publications

These publications are also relevant as further information sources:

- *CMS Application Multitasking*, SC24-5766
- *CMS Application Development Guide*, SC24-5761
- *VM/ESA CPI-C User's Guide*, SC24-5595-01
- *SAA CPI-C Reference*, SC26-4399-08

How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at http://www.redbooks.ibm.com/.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Redbooks Web Site on the World Wide Web**

  http://w3.itso.ibm.com/

- **PUBORDER** — to order hardcopies in the United States
- **Tools Disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLCAT REDPRINT
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get BookManager BOOKs of redbooks, type the following command:

  ```
  TOOLCAT REDBOOKS
  ```

  To get lists of redbooks, type the following command:

  ```
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  ```

  To register for information on workshops, residencies, and redbooks, type the following command:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1998
  ```

- **REDBOOKS Category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL

Redpieces

For information so current it is still in the process of being written, look at "Redpieces" on the Redbooks Web Site (http://www.redbooks.ibm.com/redpieces.html). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications Publications Customer Support P.O. Box 29570 Raleigh, NC 27626-0570 USA	IBM Publications 144-4th Avenue, S.W. Calgary, Alberta T2P 3N5 Canada	IBM Direct Services Sortemosevej 21 DK-3450 Allerød Denmark

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1)001-408-256-5422 (Outside USA)** — ask for:

Index # 4421 Abstracts of new redbooks
Index # 4422 IBM redbooks
Index # 4420 Redbooks for last six months

- **On the World Wide Web**

Redbooks Web Site	http://www.redbooks.ibm.com/
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

Redpieces

For information so current it is still in the process of being written, look at ″Redpieces″ on the Redbooks Web Site (`http://www.redbooks.ibm.com/redpieces.html`). Redpieces are redbooks in progress; not all redbooks become redpieces, and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number
- Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Glossary

A

abend. Abnormal end of task.

accept. In a VTAM application program, to establish a session with a logical unit (LU) in response to a CINIT request from a system services control point (SSCP). The session-initiation request may begin when a terminal user logs on, a VTAM application program issues a macroinstruction, or a VTAM operator issues a command. See also acquire.

access. (1) The manner in which files or data sets are referred to by the computer. (2) To obtain data from or to put data in storage. (3) In computer security, a specific type of interaction between a subject and an object that results in the flow of information from one to the other.

action. In a conceptual schema language, one or more elementary actions that, as a unit, change a collection of sentences into another one or make known a collection of sentences present in the information base or conceptual schema.

activate. (1) To put a device into an operational state. (2) To pass control to a program, procedure, or routine. (3) To make a resource ready to perform its function. Contrast with deactivate.

activity. The percentage of records in a file that are processed in a run.

address constant. A value, or an expression representing a value, used in calculating storage addresses.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

allocate. (1) To assign a resource, such as a disk or a diskette file, to perform a task. Contrast with deallocate. (2) A logical unit (LU) 6.2 application program interface (API) verb used to assign a session to a conversation for the conversation′s use. Contrast with deallocate.

append. A function or mode that enables a user to add a new document or character string to the end of previously entered text.

application. (1) The use to which an information processing system is put; for example, a payroll application, an airline reservation application, a network application. (2) A collection of software components used to perform specific types of user-oriented work on a computer.

assembler. A computer program that converts assembly language instructions into object code.

attention (ATTN). An occurrence external to an operation that could cause an interruption of the operation.

authority. The right to access objects, resources, or functions.

authorization. (1) In computer security, the right granted to a user to communicate with or make use of a computer system. (2) The process of granting a user either complete or restricted access to an object, resource, or function.

B

binding. (1) In programming, an association between a variable and a value for that variable that holds within a defined scope. The scope may be that of a rule, a function call or a procedure invocation. (2) In the AIX operating system, a temporary association between a client and both an object and a server that exports an interface to the object. A binding is meaningful only to the program that sets it and is represented by a bound handle.

blocking. (1) The process of combining two or more records in one block. (2) The process of combining incoming messages in a single message.

bypass. To eliminate a station or an access unit from a ring network by allowing the data to flow in a path around it.

C

calling program. A program that requests execution of another program (a called program).

check. (1) An error condition. (2) To look for a condition.

clause. (1) In COBOL, an ordered set of consecutive COBOL character-strings whose purpose is to specify an attribute of an entry. See data clause, environment clause, file clause, report clause. (2) In SQL, a distinct part of a statement in the language structure, such as a SELECT clause or a WHERE clause.

cleanup. In SNA products, a network services request, sent by a system services control point (SSCP) to a logical unit (LU), that causes a particular LU-LU session with that LU to be ended immediately without requiring the participation of either the other LU or its SSCP.

client. (1) A user. (2) A functional unit that receives shared services from a server. (3) In an AIX distributed file system environment, a system that is dependent on a server to provide it with programs or access to programs.

close. The function that ends the connection between a file and a program, and ends the processing. Contrast with open.

CMS nucleus. The portion of CMS that usually resides in the user′s virtual storage when CMS is executing. Each CMS user receives a copy of the CMS nucleus at initial program load (IPL) of CMS.

code level. The number of bits used to represent a character.

command interpreter. In the AIX operating system, a program that sends instructions to the kernel.

command line. On a display screen, a display line usually at the bottom of the screen, in which only commands can be entered.

compact. Synonym for compress.

compatibility. The capability of a hardware or software component to conform with the interface requirements of a given data processing system without adversely affecting its functions.

compile. To translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language.

configure. To describe to a system the devices, optional features, and programs installed on the system.

connectivity. (1) The capability of a system or device to be attached to other systems or devices without modification. (2) The capability to attach a variety of functional units without modifying them.

console. A part of a computer used for communication between the operator or maintenance engineer and the computer.

conversation. A dialog between a user and an interactive data processing system.

convert. To change the representation of data from one form to another, without changing the information they convey; for example, radix conversion, code conversion, analog to digital conversion, media conversion.

D

data collection. The process of bringing data together from one or more sources.

data queue. An object that is used to communicate and store data used by several programs in a job or between jobs.

debugging. Acting to detect and correct errors in software or system configuration.

default value. A value assumed when no value has been specified. Synonymous with assumed value.

deletion. (1) The removal of data from storage. (2) In a conceptual schema language, the removal of a previously inserted sentence from the information base or conceptual schema.

directory. An index that is used by a control program to locate one or more blocks of data that are stored in separate areas of a data set in direct access storage.

E

edit. (1) To add, change, delete, or rearrange data and to perform operations such as code conversion and zero suppression. (2) To enter, modify, or delete data. (3) To modify a numeric field for output by suppressing zeros and inserting commas, periods, currency symbols, the sign status, or other constant information. (4) To alter or refine information, especially text and illustrations, for publication or display.

enable. (1) To make functional. (2) In interactive communications, to load and start a subsystem. Contrast with disable.

enter. (1) To place on the line a message to be transmitted from a terminal to the computer. Contrast with accept. See also receive, send. (2) To type in information on a keyboard and press the Enter key to send the information to the computer.

entry point (EP). (1) The address or label of the first instruction executed on entering a computer program, routine, or subroutine. A computer program, routine, or subroutine may have a number of different entry points, each perhaps corresponding to a different function or purpose. Synonymous with entrance, entry. (2) In a routine, any place to which control can be passed.

error. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

event queue. In computer graphics, a queue that records changes in input devices such as buttons,

valuators, and the keyboard. The event queue provides a time-ordered list of input events.

exception. (1) In programming languages, an abnormal situation that may arise during execution, that may cause a deviation from the normal execution sequence, and for which facilities exist in a programming language to define, raise, recognize, ignore, and handle it; for example, (ON-) condition in PL/I. (2) An abnormal condition such as an I/O error encountered in processing a data set or a file.

execution time. (1) Any instant at which the execution of a particular computer program takes place. (2) The amount of time needed for the execution of a particular computer program.

exit program. Synonym for exit routine.

external function. A function supplied by the compiler when the function is referred to by name in a program. Contrast with intrinsic function.

F

facility. (1) An operational capability, or the means for providing such a capability. (2) In Open Systems Interconnection architecture, a part of a service provided by specific layer. (3) A service provided by an operating system for a particular purpose; for example, the checkpoint/restart facility.

file. (1) A named set of records stored or processed as a unit. (2) A collection of information treated as a unit. (3) A collection of related data that is stored and retrieved by an assigned name.

fill. (1) In a token-ring network, a specified bit pattern that a transmitting data station sends before or after transmission frames, tokens, or abort sequences to avoid what would otherwise be interpreted as an inactive or indeterminate transmitter state. (2) In computer graphics, a designated area of the screen that is flooded with a particular color.

filter. (1) A device or program that separates data, signals, or material in accordance with specified criteria. (2) In IBM personal computers, a program that reads output data from the keyboard (the standard input device), modifies the data, and writes output data to the display screen (the standard output device). (3) In the AIX operating system, a command that reads standard input data, modifies the data, and sends it to the display screen.

find. Synonym for search.

fixed. (1) In System/370 virtual storage systems, not capable of being paged-out. (2) Synonym for resident. (3) Synonym for read-only.

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of values.

G

generation. A means of referencing items with respect to time and ancestry so that an item without antecedents is designated as the first or n-th generation, and subsequent derivations are designated as n+1, n+2, and so on.

generic profile. In RACF, a resource profile that can provide protection for one or more resources. The resources protected by a generic profile have similar names and identical security requirements. For example, a generic data set profile can protect one or more data sets.

get. To obtain a record from an input file.

group. (1) A set of related records that have the same value for a particular field in all records. (2) A series of records logically joined together. (3) A series of lines repeated consecutively as a set on a full-screen form or full-screen panel. (4) A list of names that are known together by a single name. (5) In RACF, a collection of users who can share access authorities for protected resources.

H

handler. A routine that controls a program′s reaction to specific external events; for example, an interrupt handler.

hexadecimal. Pertaining to a system of numbers to the base 16; hexadecimal digits range from 0 through 9 and A through F, where A represents 10 and F represents 15.

I

immediate command. In VM, a CMS command that, when issued after an attention interruption, causes any program execution currently in progress to be suspended until the immediate command is processed. The immediate commands are HB (halt the execution of the CMS batch virtual machine at the end of current job), HO (halt SVC tracing), HT (halt typing or displaying), HX (halt execution), RO (resume tracing), RT (resume typing or displaying), and SO (suspend tracing temporarily).

index. (1) A list of the contents of a file or of a document, together with keys or references for locating the contents. (2) In programming, an integer that identifies the position of a data item in a sequence of data items. (3) A symbol or a numeral

used to identify a particular quantity in an array of similar quantities.

initialization. (1) The operations required for setting a device to a starting state, before the use of a data medium, or before implementation of a process. (2) Preparation of a system, device, or program for operation.

input. (1) Pertaining to a device, process, or channel involved in an input process, or to the associated data or states. The word ″input″ may be used in place of ″input data,″ ″input signal″, ″input process″, when such a usage is clear in a given context. (2) Pertaining to a functional unit or channel involved in an input process or to the data involved in such a process.

insert. (1) A function or mode that enables the introduction of additional characters within previously entered text. (2) To introduce data between previously stored items of data. (3) The source entry utility operation during which source statements are keyed in and added as new records in a source member.

install. To add a program, program option, or software to a system in such a manner that it is runnable and interacts properly with all affected programs in the system.

integrated. Pertaining to a feature that is part of a device. Synonymous with built-in.

interface. (1) A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. (2) Hardware, software, or both, that links systems, programs, or devices.

interpreter. (1) A computer program that can interpret. (2) A program that translates and executes each instruction of a high-level programming language before it translates and executes the next instruction.

interprocess communication. (1) In the OS/2 operating system, the exchange of information between processes or threads through semaphores, queues, and shared memory. (2) In the AIX operating system, the process by which programs communicate data to each other and to synchronize their activities. Semaphores, signals, and internal message queues are common methods of inter-process communication.

K

key. (1) An identifier within a set of data elements. (2) One or more characters used to identify the record and establish the order of the record within an indexed file. (3) In VSAM, one or more consecutive characters taken from a data record, used to identify the record and establish its order with respect to other records. See also alternate key, key field. (4) In VTAM, a character string that matches a definition in the key table. This key identifies the destination of a message or special processing to be done on that message. See also key table. (5) To enter information from a keyboard.

keyboard. A group of numeric keys, alphabetic keys, or function keys used for entering information into a terminal and into the system.

L

library. (1) A collection of functions, calls, subroutines, or other data. (2) A data file that contains files and control information that allows them to be accessed individually. (3) A named area on disk that can contain programs and related information (not files). A library consists of different sections, called library members.

line. (1) In text, a horizontal, linear arrangement of graphic characters. (2) The portion of a data circuit external to data circuit-terminating equipment (DCE), that connects the DCE to a data switching exchange (DSE), that connects a DCE to one or more other DCEs, or that connects a DSE to another DSE.

linkage. In computer security, the combination of data or information from one information system with data or information from another system in the hope of deriving additional information; for example, the combination of computer files from two or more sources.

literal. (1) A symbol or a quantity in a source program that is itself data, rather than a reference to data. (2) A character string whose value is given by the characters themselves; for example, the numeric literal 7 has the value 7, and the character literal ″CHARACTERS″ has the value CHARACTERS.

loader. (1) A routine, commonly a computer program, that reads data into main storage. (2) In the AIX operating system, a program that reads run files into main storage so that the files can be run.

local. (1) In programming languages, pertaining to the relationship between a language object and a block such that the language object has a scope contained in that block. (2) Pertaining to that which is defined and used only in one subdivision of a computer program. (3) Pertaining to a device

accessed directly without use of a telecommunication line. (4) Synonym for channel-attached.

logic. The systematized interconnection of digital switching functions, circuits, or devices.

loop. (1) A sequence of instructions that is to be executed iteratively. (2) A closed unidirectional signal path connecting input/output devices to a system. (3) Synonym for ring network.

M

mapping. (1) A list, usually in a profile, that establishes a correspondence between items in two groups; for example, a keyboard mapping can establish what character is displayed when a certain key is pressed. See also keyboard mapping. (2) In a database, the establishing of correspondences between a given logical structure and a given physical structure.

matching. The technique of comparing the keys of two or more records to select items for a particular stage of processing or to reject invalid records.

message. (1) A communication sent from a person or program to another person or program. (2) A unit of data sent over a telecommunication line.

minidisk. Synonym for virtual disk

minimize. To remove from the screen all windows associated with an application and replace them with an icon that represents the application.

module. (1) A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine. (2) A part of a program that usually performs a particular function or related functions.

monitor. (1) A device that observes and records selected activities within a data processing system for analysis. Possible uses are to indicate significant departure from the norm, or to determine levels of utilization of particular functional units. (2) Software or hardware that observes, supervises, controls, or verifies operations of a system.

multitasking. A mode of operation that provides for concurrent performance, or interleaved execution of two or more tasks.

N

native. Deprecated term for IBM-supplied, basic, required, or stand-alone.

network. (1) An arrangement of nodes and connecting branches. (2) A configuration of data processing devices and software connected for information interchange. (3) A group of nodes and the links interconnecting them.

networking. In a multiple-domain network, communication between domains. Synonymous with cross-domain communication.

node. (1) In a network, a point at which one or more functional units connect channels or data circuits. (2) In network topology, the point at an end of a branch. (3) In a tree structure, a point at which subordinate items of data originate. (4) An endpoint of a link or a junction common to two or more links in a network. Nodes can be processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities. (5) In VTAM, a point in a network defined by a symbolic name. See major node, minor node.

nucleus. That part of a control program resident in main storage. Synonymous with resident control program.

O

open. (1) The function that connects a file to a program for processing. (2) To prepare a file for processing. (3) Contrast with close. (4) In the IBM Token-Ring Network, to make an adapter ready for use.

operating system (OS). Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible.

option. A specification in a statement that may be used to influence the execution of the statement.

origin system. Synonym for input system.

originator. (1) The user who creates, addresses, and usually sends a message. (2) Contrast with recipient.

output. (1) Pertaining to a device, process, or channel involved in an output process, or to the associated data or states. The word ″output″ may be used in place of ″output data,″ ″output signal″, ″output process″, when such a usage is clear in a given context. (2) Data that has been processed. (3) Data transferred from storage to an output device.

P

parse. (1) In systems with time sharing, to analyze the operands entered with a command and create a parameter list for the command processor from the information. (2) In REXX, to split a string into parts, using function calls or by using a parsing template on the ARG, PARSE, or PULL instructions.

partner. In data communications, the remote application program or the remote computer.

pass. One cycle of processing a body of data.

performance. One of the two major factors, together with facility, on which the total productivity of a system depends. Performance is largely determined by a combination of throughput, response time, and availability.

pipeline. (1) A serial arrangement of processors or a serial arrangement of registers within a processor. Each processor or register performs part of a task and passes results to the next processor; several parts of different tasks can be performed at the same time. (2) To start execution of an instruction sequence before the previous instruction sequence is completed to increase processing speed.

pool. A division of main or auxiliary storage.

port. (1) An access point for data entry or exit. (2) A specific communications end point within a host. A port is identified by a port number. (3) The representation of a physical connection to the link hardware. A port is sometimes referred to as an adapter; however, there can be more than one port on an adapter. There may be one or more ports controlled by a single DLC process. (4) To make the programming changes necessary to allow a program that runs on one type of computer to run on another type of computer.

prefix. (1) A code dialed by a caller before being connected. Contrast with suffix. (2) A code at the beginning of a message or record.

preprocessing. Processing for a display that occurs before the display is shown.

priority. (1) A rank assigned to a task that determines its precedence in receiving system resources. (2) The relative significance of one job to other jobs in competing for allocation of resources.

privilege class. In VM, one or more classes assigned to virtual machine user in the user′s VM directory entry; each privilege class specified allows access to a logical subset of all the CP commands.

processing. The performance of logical operations and calculations on data, including temporary retention of data in processor storage while the data is being operated on.

profile. (1) Data that describes the significant characteristics of a user, a group of users, or one or more computer resources. (2) In computer security, a description of the characteristics of an entity to which access is controlled. (3) A description of the control available to a particular network operator.

programming. The design, writing, modifying, and testing of programs.

prompt. A visual or audible message sent by a program to request the user′s response.

Q

queue. (1) A list constructed and maintained so that the next data element to be retrieved is the one stored first. (2) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed.

R

reason code. A code that identifies the reason for a detected error.

receiver. A person or thing that receives something. Contrast with sender.

receiving. In VTAM, the process by which the host processor obtains a message entered at a station. Contrast with sending.

remote access. Pertaining to communication with a data processing facility through a data link.

remote system. Any other system in a network with which a system can communicate.

request. (1) A directive, by means of a basic transmission unit, from an access method that causes the network control program to perform a data-transfer operation or auxiliary operation. (2) In SNA, a message unit that signals initiation of an action or protocol; for example, INITIATE SELF is a request for activation of an LU-LU session.

resource. (1) Any of the data processing system elements needed to perform required operations, including storage, input/output units, one or more processing units, data, files, and programs. Synonymous with computer resource. (2) Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, processing unit, data sets, and control or processing programs.

return code. (1) A code used to influence the execution of succeeding instructions. (2) A value

returned to a program to indicate the results of an operation requested by that program.

S

scope. (1) The portion of an expression to which the operator is applied. (2) The portion of a computer program within which the definition of the variable remains unchanged.

search. (1) A function or mode that enables the user to locate occurrences of such things as particular character strings, embedded commands, or boldface letters in a document. Synonymous with find. (2) The process of looking for a particular item. See also browse, scan. (3) To scan one or more data elements of a set in order to find elements that have a certain property.

semaphore. (1) A variable that is used to enforce mutual exclusion. (2) An indicator used to control access to a file; for example, in a multiuser application, a flag that prevents simultaneous access to a file. (3) An entity used to control access to system resources. Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

sending. In VTAM, the process by which the host processor places a message on a line for transmission to a station. Contrast with receiving.

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server. (3) In the AIX operating system, an application program that usually runs in the background and is controlled by the system program controller. (4) In TCP/IP, a system in a network that handles the requests of a system at another site, called a client-server.

service. (1) In Open Systems Interconnection architecture, a capability of a given layer and the layers below it that is provided to the next higher layer. The service of a given layer is provided at the boundary between this layer and the next higher layer. (2) A customer-related or product-related business function such as design/manufacturing error correction, installation planning, maintenance, customer education, or programming assistance.

session. (1) In network architecture, for the purpose of data communication between functional units, all the activities which take place during the establishment, maintenance, and release of the connection. (2) A logical connection between two network accessible units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. Each session is uniquely identified in a transmission header (TH) accompanying any transmissions exchanged during the session. (3) The period of time during which a user of a terminal can communicate with an interactive system, usually, elapsed time between logon and logoff.

setup. (1) In a computer that consists of an assembly of individual computing units, the arrangement of connections between the units, and the adjustments needed for the computer to operate on a problem. (2) The preparation of a computing system to perform a job or job step. Setup is usually performed by an operator and often involves performing routine functions, such as mounting tape reels and loading card decks.

shared file. A file that can be used by two computers or two systems at the same time. A shared file can link two systems.

shutdown. The process of ending operation of a system or a subsystem, following a defined procedure.

SNA network. The part of a user-application network that conforms to the formats and protocols of Systems Network Architecture. It enables reliable transfer of data among end users and provides protocols for controlling the resources of various network configurations. The SNA network consists of network accessible units (NAUs), boundary function, gateway function, and intermediate session routing function components; and the transport network.

socket. (1) In the AIX operating system: (a) A unique host identifier created by the concatenation of a port identifier with a transmission control protocol/Internet protocol (TCP/IP) address. (b) A port identifier. (c) A 16-bit port number. (d) A port on a specific host; a communications end point that is accessible through a protocol family′s addressing mechanism. A socket is identified by a socket address. (2) The abstraction provided by the University of California′s Berkeley Software Distribution (commonly called Berkeley UNIX or BSD UNIX) that serves as an endpoint for communication between processes or applications.

software. (1) All or part of the programs, procedures, rules, and associated documentation of a data processing system. Software is an intellectual creation that is independent of the medium on which it is recorded. (2) Contrast with hardware.

subroutine. (1) A sequence of instructions whose execution is invoked by a call. (2) A sequenced set of instructions or statements that may be used in one or more computer programs and at one or more points in a computer program. (3) A group of instructions that can be part of another routine or can be called by another program or routine.

subset. (1) A set each element of which is an element of a specified other set. (2) A variant form of a programming language with fewer features or more restrictions than the original language. (3) A set contained within a set.

symbol. (1) A graphic representation of a concept that has meaning in a specific context. (2) A representation of something by reason of relationship, association, or convention.

synchronization. The action of forcing certain points in the execution sequences of two or more asynchronous procedures to coincide in time.

syntax. (1) The relationship among characters or groups of characters, independent of their meanings or the manner of their interpretation and use. (2) The rules governing the structure of a language.

system. In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions.

T

tag. One or more characters attached to a set of data that contain information about the set, including its identification.

target. (1) Pertaining to a storage device to which information is written. (2) The program or system to which a request is sent. (3) The location to which the information is destined. (4) A system, program, or device that interprets, rejects or satisfies, and replies to requests received from a source.

terminate. (1) In SNA products, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure for ending one or more designated LU-LU sessions. (2) To stop the operation of a system or device. (3) To stop execution of a program.

testing. The running of a system or a program against a predetermined series of data to arrive at a predictable result for the purpose of establishing the acceptability of the system or program.

thread. (1) In the OS/2 operating system, the smallest unit of operation to be performed within a process. (2) A link between an IMS/VS subsystem and a Database 2 (DB2) subsystem; resources in the external DB2 subsystem are allocated to that link or thread.

timeout. (1) An event that occurs at the end of a predetermined period of time that began at the occurrence of another specified event. (2) A time interval allotted for certain operations to occur; for example, response to polling or addressing before system operation is interrupted and must be restarted.

token. (1) In a local area network, the symbol of authority passed successively from one data station to another to indicate the station temporarily in control of the transmission medium. Each data station has an opportunity to acquire and use the token to control the medium. A token is a particular message or bit pattern that signifies permission to transmit. (2) A sequence of bits passed from one device to another along the token ring. When the token has data appended to it, it becomes a frame. (3) In a programming language, a character string, in a particular format, that has some defined significance.

trace. (1) A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (2) The process of recording the sequence in which the statements in a program are executed and, optionally, the values of the program variables used in the statements.

transfer. (1) To send data from one place and receive the data at another place. Synonymous with move. (2) To read data from auxiliary storage or from an input device into processor storage or from processor storage to auxiliary storage or to an output device.

tutorial. Information presented in a teaching format.

U

uppercase. Pertaining to the capital letters, as distinguished from the small letters; for example, A, B, G, rather than a, b, g.

user. (1) A person who requires the services of a computing system. (2) Any person or any thing that may issue or receive commands and messages to or from the information processing system.

user ID. User identification.

V

variable. In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type.

verify. To determine whether a transcription of data or other operation has been accomplished accurately.

virtual machine (VM). A virtual data processing system that appears to be at the exclusive disposal of a particular user, but whose functions are accomplished by sharing the resources of a real data processing system.

W

work area. An area reserved for temporary storage of data that are to be operated on.

writing. The action of making a permanent or transient recording of data in a storage device or on a data medium.

Z

zero. In data processing, the number that, when added to or subtracted from any other number, does not alter the value of the other number. Zero may have different representations in computers, such as positively or negatively signed zero (which may result from subtracting a signed number from itself) and floating-point zero (in which the fixed point part is zero while the exponent in the floating-point representation may vary).

List of Abbreviations

APAR	Authorized Program Analysis Report
APPC	Advanced Program-to-Program Communication
AVS	APPC/VM VTAM Support
BG	BackGround
CMS	Conversational Monitor System
CP	Control Program
CPI	Callable Programming Interface
CPI-C	Common Programming Interface for Communications
CSL	Callable Services Library
IBM	International Business Machines Corporation
ID	IDentification/IDentifier
IP	Internet Protocol
IPC	Inter-Processor Communication
IPL	Initial Program Load
ITSO	International Technical Support Organization
IUCV	Inter-User Communications Vehicle
LIFO	Last In/First OutLast In/First Out
MT	MultiTasking
PC	Personal Computer
POSIX	Portable Operating System Interface for Computer Environments
PTF	Program Temporary Fix
RACF	Resource Access Control Facility
REXX	REstructured eXtended eXecutor Language
RSCS	Remote Spooling Communications Subsystem
RXSOCKET	REXX socket function
SCIF	Single Console Image Facility
SFS	Shared File System
SNA	Systems Network Architecture
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TSAF	Transparent Services Access Facility
UCR	Unsatisfactory Condition Report
URL	Uniform Resource Locator
VM	Virtual Machine
VM/ESA	Virtual Machine/Enterprise Systems Architecture
XA	Extended Architecture
XC	Extended Configuration

Index

Special Characters

A

B

C

D

E

G

I

K

M

N

O

P

Q

R

S

T

U

V

ITSO Redbook Evaluation

Exploiting Recent CMS Function: A User's Guide to CMS Application Multitasking
SG24-5164-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. **Please complete this questionnaire and return it using one of the following methods:**

- Use the online evaluation form found at http://www.redbooks.ibm.com
- Fax this form to: USA International Access Code 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
__Customer __Business Partner __Solution Developer __IBM employee
__None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction _____

Please answer the following questions:

Was this redbook published in time for your needs? Yes____ No____

If no, please explain:

_____

_____

_____

_____

What other redbooks would you like to see published?

_____

_____

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

_____

_____

_____

_____

_____

SG24-5164-00
Printed in the U.S.A.

Artwork Definitions

Table Definitions

Figures

Headings

REDBIBM	REDB$ORD	109	How IBM Employees Can Get ITSO Redbooks
REDBCUS	REDB$ORD	110	How Customers Can Get ITSO Redbooks
REDBFOR	REDB$ORD	111	IBM Redbook Order Form
REDBEVA	REDB$EVA	127	ITSO Redbook Evaluation

Index Entries

id	File	Page	References
TREDIND	5164VARS	i	(1) thread 1, 41, 46, 48, 99, 100, 100

Processing Options

Runtime values:

Document fileid	SG245164 SCRIPT
Document type	USERDOC
Document style	REDBOOK
Profile	EDFPRF40
Service Level	0022
SCRIPT/VS Release	4.0.0
Date	98.11.20
Time	05:14:56
Device	3820A
Number of Passes	4
Index	YES
SYSVAR D	YES
SYSVAR G	INLINE
SYSVAR X	YES

Formatting values used:

Annotation	NO
Cross reference listing	YES
Cross reference head prefix only	NO
Dialog	LABEL
Duplex	YES
DVCF conditions file	(none)
DVCF value 1	(none)
DVCF value 2	(none)
DVCF value 3	(none)
DVCF value 4	(none)
DVCF value 5	(none)
DVCF value 6	(none)
DVCF value 7	(none)
DVCF value 8	(none)
DVCF value 9	(none)
Explode	NO
Figure list on new page	YES
Figure/table number separation	YES
Folio-by-chapter	NO
Head 0 body text	Part
Head 1 body text	Chapter
Head 1 appendix text	Appendix
Hyphenation	NO
Justification	NO
Language	ENGL
Keyboard	395
Layout	OFF
Leader dots	YES
Master index	(none)
Partial TOC (maximum level)	4
Partial TOC (new page after)	INLINE
Print example id's	NO
Print cross reference page numbers	YES
Process value	(none)
Punctuation move characters	.,
Read cross-reference file	(none)
Running heading/footing rule	NONE
Show index entries	NO
Table of Contents (maximum level)	3
Table list on new page	YES

Title page (draft) alignment ..... RIGHT
Write cross-reference file ..... (none)

Imbed Trace

Page 0	5164SU
Page 0	5164VARS
Page 0	REDB$POK
Page i	REDB$ED1
Page i	5164EDNO
Page i	REDB$ED2
Page ix	5164ABST
Page ix	5164ACKS
Page x	REDB$COM
Page x	5164IMBD
Page x	5164CH01
Page 3	5164CH02
Page 28	5164CH03
Page 50	5164CH04
Page 94	5164AX01
Page 97	5164AX02
Page 101	5164AX03
Page 105	5164SPEC
Page 105	REDB$SPE
Page 105	5164TMKS
Page 106	5164BIBL
Page 107	REDB$BIB
Page 108	REDB$ORD
Page 111	5164GLOS
Page 121	5164ABRV
Page 126	REDB$EVA